UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Blaž Peruš

Napredne tehnologije v razvoju spletnih aplikacij

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Mira Trebar

Ljubljana, 2016

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Napredne tehnologije v razvoju spletnih aplikacij

Tematika naloge:

Kandidat naj v diplomskem delu pregleda in analizira napredne tehnologije, ki se uporabljajo pri načrtovanju in izdelavi spletnih aplikacij. Na praktičnem primeru prodaje in nakupa rabljenih vozil naj predstavi ustrezne metodologije in principe hitrega in agilnega razvoja. Predstavljena rešitev naj vključuje še druge pomembne pristope in orodja, ki omogočajo uspešen razvoj zagonskega podjetja.

Iz srca se zahvaljujem vsem, ki jih bom omenil v naslednjem odstavku. Ne le za podporo pri študiju, ampak tudi za nasvete, lekcije, razumevanje in čas zunaj študija. Staršema, ki sta me in me še vedno močno podpirata kljub moji trmi pri večni izbiri drugačnih in ponavadi težjih poti. Starim staršem na obeh straneh, ki so me podpirali z manj pritiska in z bolj pozitivno energijo, kar je pripomoglo k bolj sproščenim in jasnim vizijam. Sestri, ki me je naučila, da je življenje krhko in da moramo živeti vsak dan z vso močjo. Lei, ker je s svojo prisotnostjo dodala ženski in manj introvertiran pogled na življenje in pripomogla k redefiniranju moje lastne definicije uspeha. Mateju B., ki je brezpogojno pomagal v najpomembnejših trenutkih. Mateju M., ki je vedno poskrbel, da ni bila nobena ovira videti prevelika, in me je naučil, da je vsaka resnica bolje sprejeta, če je povedana s humorjem. Aljažu za razum, usmeritev, mentorstvo in motivacijo neprecenljive dodane vrednosti. Nazadnje se iz srca zahvaljujem tudi mentorici doc. dr. Miri Trebar, ki je z izjemno potrpežljivostjo, nasveti in usmeritvijo pomagala, da sem pravočasno in uspešno napisal diplomo.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	Application programming interface	Vmesnik uporabniškega programa
ASD	Adaptive Software Development	Prilagodljiv razvoj programske opreme
CDN	Content Delivery Network	Sistem za hitro podajanje spletne vsebine
CSS	Cascade Style Sheets	Kaskadne stilske predloge
DB	Database	Podatkovna baza
FTP	File Transfer Protocol	Protokol prenosa datotek
GA	Google Analytics	Googlova analitična orodja
HTML	Hyper Text Markup Language	Jezik za označevanje nadbesedila
HTTP	Hypertext Transfer Protocol	HTTP protokol
HTTPS	Hypertext Transfer Protocol over SSL	HTTPS protokol
IMAP	Internet Message Access Protocol	IMAP protokol
MVC	Model-View-Controler	Model-Pogled-Kontroler
OS	Operating System	Operacijski sistem
PDF	Portable Document Format	PDF datoteka
PITR	Point-in-time recovery	Obnovitev iz točke v času
POP3	Post Office Protocol version 3	POP3 protokol
RoR	Ruby on Rails	Ruby on Rails
SEO	Search Engine Optimization	Optimizacija za spletne brskalnike
SMTP	Simple Mail Transfer Protocol	SMTP protokol
SQL	Structured Query Language	Strukturirani povpraševalni jezik
SSH	Secure Shell	SSH protokol
SSL	Secure Socker Layer	SSL protokol
USA	United States of America	Združene države Amerike

Povzetek

Naslov: Napredne tehnologije v razvoju spletnih aplikacij

Cilj diplomske naloge je predstavitev razvoja spletnega projekta Autocommerce in opis uporabljenih tehnologij. Na podlagi tega pa predstaviti tudi metodologije agilnega razvoja, s pomočjo katerih je možno hitro in učinkovito postaviti celoten informacijski sistem od začetne ideje do končne prve verzije. Takšne metodologije so danes uporabljene predvsem pri gradnji zagonskih podjetij. Tako je mogoče v zelo kratkem času razviti in testirati produkte, ki jih je možno prilagajati na podlagi izbranih odzivov, kar omogoča hitro odzivanje pri iskanju optimalne pozicije na tržišču. Spletna aplikacija Autocommerce je namenjena povezovanju kupcev in prodajalcev rabljenih vozil, ki so pregledana in preverjena s strani specializirane pristojne organizacije. Takšen pristop omogoča, da kupec hitreje najde primerno vozilo in ima ob tem zagotovilo, da bo brskal med dobro dokumentiranimi vozili. Na podlagi boljših in preverjenih informacij se hitreje odloči za nakup. Opisan je tudi začetni poslovni model in proces zbiranja analitičnih podatkov v aplikaciji. V diplomski nalogi smo kot rešitev razvili spletno aplikacijo s pomočjo ogrodja Ruby on Rails in po korakih predstavili razvoj vsakega dela aplikacije. Opisali in predstavili smo tudi zunanja orodja, ki olajšajo delo in dodajo funkcionalnosti. Predvsem pa omogočajo povezavo vseh delov v eno celoto, ki ji pravimo informacijski sistem.

Ključne besede: spletni projekt, spletna aplikacija, Ruby on Rails, Scrum, rabljena vozila, analitika

Abstract

Title: Advanced technologies in web application design

The main goal of this diploma thesis is a presentation of Autocommerce web project development. All used technologies are described. On the basis of the project we also present agile methodologies for software development which helps us to efficiently build the whole information system from the idea to the first version. Such methodologies are mostly used by startups, because they are perfect for building and testing products in a very short time & for adapting with the provided feedback from users. With process like that it's possible to achieve the optimal position on the market. Web application Autocommerce connects buyers and sellers of used & verified vehicles. All vehicles in the application are verified by specialized organisation. Like that users spend less time searching for a new vehicle because they know the condition of a vehicle. The thesis also describes business model of the application and how it collects analytics data inside the application. The end product of this diploma is a web application developed with Ruby on Rails framework. All major parts of an applications are described with short examples, as well as external tools and services that extend the functionalities and helps us develop it even further. They are important, because they connect all ends of the application into one information system.

Keywords: web project, web application, Ruby on Rails, Scrum, secondhand vehicles, analytics

1. Uvod

Motivacija za pisanje diplome o takšnem projektu je nastopila ob vprašanju “Kakšno diplomsko nalogo lahko napišemo, da bomo predstavili nekaj pomembnega znanja o spletnih tehnologijah, ki bi si ga želeli prebrati sami ob vstopu na fakulteto?”. Poleg takšnega vprašanja se nam postavlja še mnogo drugih. Zato se tudi osredotočamo na razvoj različnih delov aplikacije in na tesno povezavo z zunanjimi orodji, ki jih aplikacija potrebuje za uspešen zagon in delovanje, ne pa samo na programerski razvoj in kodo. V primeru, da takšna aplikacija postane uspešna, se iz nje lahko razvije zagonsko podjetje.

Namen takšnega pisanja in predstavitve je pregled nad delovanjem in vzpostavitvijo informacijskega sistema in potrebna miselna usmeritev za njegovo hitro izvedbo. Opis razvoja je namenjen predstavitvi učnega projekta, s kakršnimi se naučimo principov, ki jih lahko uporabljamo na vseh področjih uporabe spletnih tehnologij. Zelo dober primer tega je uporaba že razvitih paketov in zunanjih orodij, ki nam poenostavijo delo in izboljšajo uspeh aplikacije.

V opisu procesa razvijanja aplikacije se ne osredotočamo samo na programski razvoj, ampak tudi na druge dele razvoja, ki so pomembni pri praktičnem delu. S tem želimo vključiti celoten spekter storitev in orodij, ki jih danes potrebuje spletna aplikacija za delovanje, preboj na trg in obstoj. Dober primer takšne storitve je analitika, s katero lahko predvidevamo razvoj aplikacije v prihodnosti in upoštevamo neposreden odziv uporabnika. Tako na podlagi konkretnih povratnih informacij usmerjamo razvoj in bolj natančno razvijamo aplikacijo, ki jo bodo uporabniki želeli uporabljati.

V diplomski nalogi bomo prikazali izdelavo spletne aplikacije in orodja ter načine, po katerih smo aplikacijo izdelali. Opisali bomo različne dele aplikacije in posameznih storitev. V nadaljevanju bomo predstavili podroben razvoj posameznih delov aplikacije in našteli uporabljena orodja ter utemeljili njihovo uporabo.

Namen spletne aplikacije je povezovanje prodajalcev in kupcev preverjenih vozil. Kupci v drugih aplikacijah brskajo in kupujejo med nepreverjenimi vozili, ker so jih na spletne strani dodali kupci v želji po hitri prodaji. Tako so nekatere informacije o vozilih prikrite ali popačene. Ker smo se temu želeli izogniti, smo se odločili, da bomo vsa vozila pred dodajanjem na spletno stran pregledali. Preglede bo tako izvajala avtorizirana in

specializirana organizacija, ki bo vsako pregledano vozilo vnesla v spletno aplikacijo skupaj z uradnim potrdilom o pregledu in celotnim opisom vozila in njegovih napak.

Za izdelavo spletne aplikacije smo se odločili, ker želimo nadgraditi osnovni model prodaje rabljenih vozil in smo na tržišču prepoznali potrebo po izboljšanju obstoječih načinov. V Nemčiji se od zavarovalnic in izdelovalcev avtomobilov proda okoli 2500 avtomobilov na teden, ki jih kupijo preprodajalci avtomobilov. Do teh vozil navaden kupec nima dostopa. Do baz teh podatkov želimo dostopati neposredno brez preprodajalcev in jih učinkovito in za boljšo ceno direktno prodati končnim kupcem.

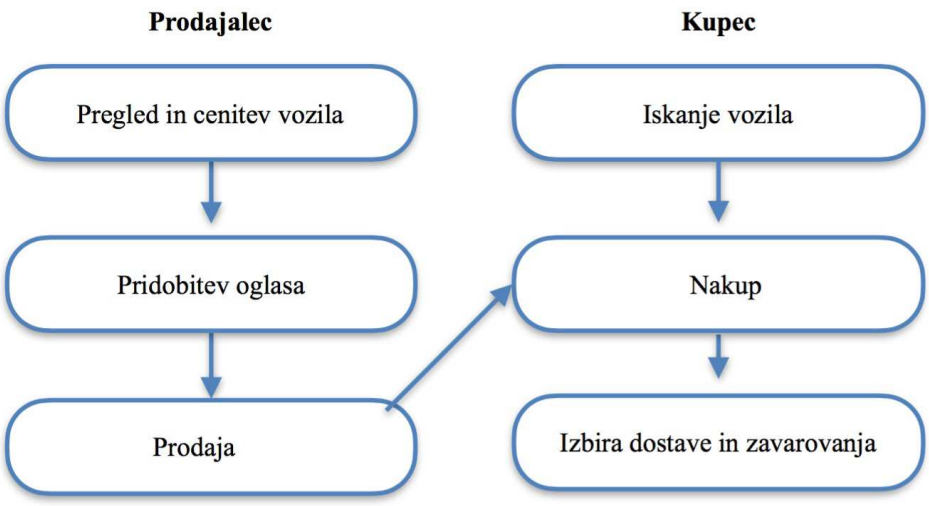
V diplomski nalogi smo za naročnika iz Nemčije dodelali poslovni načrt in razvili aplikacijo Autocommerce.de, ki je v procesu konstantnega razvijanja, zato so možna odstopanja med izgledom aplikacije na spletu ter diplomsko nalogo. Prav tako v času pisanja partnerstvo z organizacijo za pregledovanje avtomobilov še ni bilo vzpostavljeno, zato so nekateri procesi samo opisani.

2. Predstavitev spletne aplikacije

Z natančnejšim opisom izdelave spletne aplikacije smo želeli v diplomski nalogi povezati programerski razvoj in postavitev informacijskega sistema. Postavitev informacijskega sistema zajema načrtovanje pri izbiri spletnih tehnologij, uporabo zunanjih orodij, poslovni načrt, marketing in agilne metodologije. Vse našteto je med seboj močno prepleteno. Slednje je zelo pomembno, saj se tudi diplomska naloga navezuje predvsem na povezovanje različnih orodij, razvoja in principov hitrega in agilnega razvoja. Takšen pristop danes uporabljajo zagonska podjetja in je uveljavljen kot moderen način razvoja, ki se je razvil skupaj z množico zagonskih podjetij.

2.1 Koncept

Zasnova aplikacije je relativno preprosta. Želimo povezati prodajalce in kupce rabljenih vozil. Vozila, ki bodo objavljena v aplikaciji, bodo strokovno pregledana. Preglede bo izvajal partner oziroma podjetje, ki bo imelo pooblastila za izvajanje ustreznih storitev. Ob plačljivem pregledu vozila bo prodajalec pridobil uporabniški račun, ki omogoča prodajo vozila. Registracija za kupce pa je brezplačna. Slika 1 prikazuje zelo poenostavljen potek prodaje vozila v aplikaciji. Prodajalčev prvi korak je, da partnerju (ki v tej fazi še ni določen) za preglede in cenitev dostavi vozilo, ki ga bo pregledal. Ob koncu pregleda se v aplikacijo vnesejo vsi podatki o vozilu. Uporabnik nato pridobi uporabniški račun za prodajalca in oglas na spletni strani. Potem lahko začne prodajati svoje vozilo

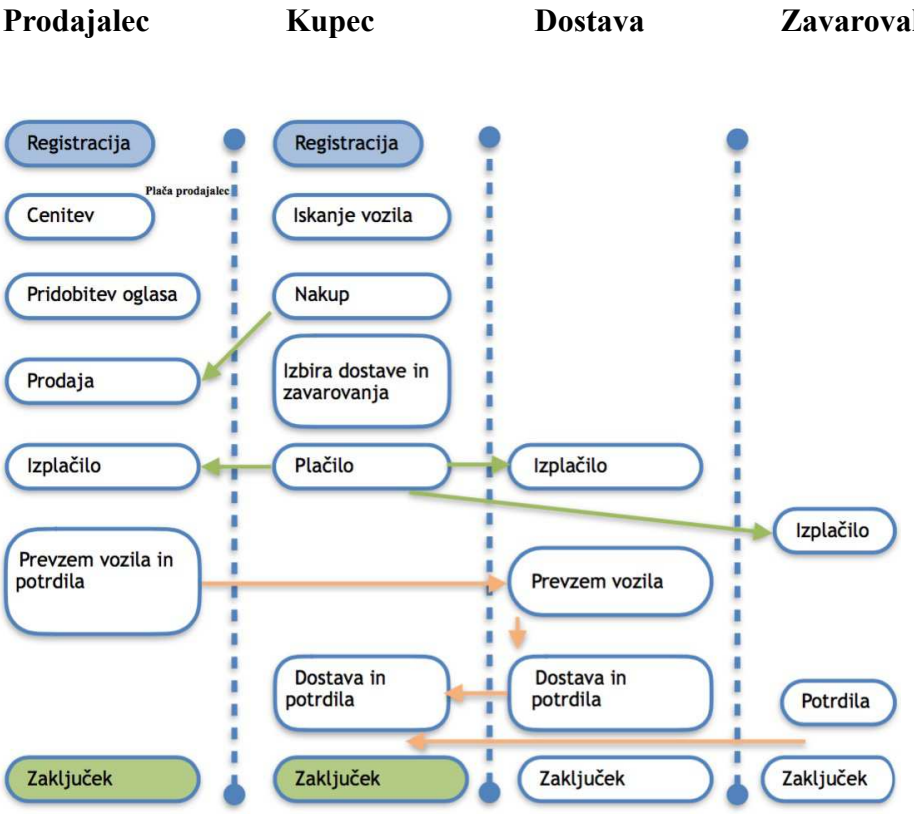


Slika 1: Prikaz najbolj enostavnega delovanja aplikacije

Kupec na drugi strani začne z brezplačno registracijo in z iskanjem vozila. Ko najde primerno vozilo in se odloči za nakup, mu ponudimo možnost dostave in primerno zavarovanje za vozilo. Tukaj se nakup vozila zaključi.

2.2 Podrobna zasnova in poslovni načrt

Začetni poslovni načrt je bil nujen, ker je narekoval določene dele razvoja aplikacije, ki so tesno povezani s celotno zasnovo. V poslovnem načrtu smo definirali našo ciljno skupino uporabnikov, probleme, ki jih s spletno aplikacijo rešujemo, ter rešitev, ki jo ponujamo. Definirali smo način pridobivanja uporabnikov v začetni fazi razvoja in sestavili načrt, kako bomo z njihovo pomočjo testirali aplikacijo. Določili in standardizirali smo tudi kratek opis ter enostavčno poved, ki natančno opisuje aplikacijo in njen namen. Slika 2 opisuje proces, ki poteka pri prodaji vozila. Proces poteka navzdol. Vsak okvir opisuje trenutno fazo, v kateri je uporabnik oziroma entiteta. Puščice prikazujejo smer izvedbe nakupa ali prodaje v obliki transakcije. Del vsake denarne transakcije znotraj aplikacije se zaračuna. Partner za cenitev in pregled vozil ni prikazan v grafu, ker cenitev plača prodajalec vozila neposredno partnerju za cenitev.



Slika 2: Proces prodaje vozila

Entitete so razdeljene tako, da je v prvem stolpcu prodajalec. V drugem je kupec in v tretjem je partner, ki skrbi za dostavo vozil. V zadnjem stolpcu je zavarovalnica, ki skrbi za zavarovanje sklenjenih nakupov in zavarovanje vozil.

Ciljni uporabniki

Ciljni uporabniki aplikacije so skupina ljudi, ki se ukvarja s prodajo in nakupom rabljenih avtomobilov. Razširjena ciljna skupina so vsi ljudje, ki kupujejo rabljene avtomobile preko spleta. Nazadnje so tukaj še ljudje, ki občasno obiskujejo strani, specializirane za avtomobilizem.

Rešitev

Za ciljne uporabnike smo sestavili sistem prodaje in nakup avtomobilov, ki vsebuje preverjanje, zavarovanje ter dostavo avtomobilov. Deluje tako, da prodajalec s preverjanjem avtomobila pri avtoriziranem partnerju preveri stanje avtomobila, kjer se opravi tudi cenitev, ter s tem dejanjem pridobi oglas v naši aplikaciji. Oglasa ne vnese kupec, ampak partner, ki skrbi za preglede. S tem se izognemo napačnim in nepreverjenim informacijam. Prodajalec nato nadzira svoj oglas in prodaja avtomobil.

Kupec, ki brska po aplikaciji, ima možnost nakupa avtomobila z dodatnim zavarovanjem in dostavo na dom. S tem prihrani čas in denar, ki bi ga porabil z ogledom in ocenjevanjem stanja avtomobila in nazadnje tudi s prevozom do doma. Ker so avtomobili preverjeni in ocenjeni, ima kupec večje zaupanje v avtomobil. Pri nakupu ima kupec možnost sklenitve zavarovanja za kupljeni avtomobil.

Zaslužek z aplikacijo

Vsak uporabnik se lahko zastonj registrira in brska po aplikaciji. Stroške preverjanja avtomobilov krijejo prodajalci avtomobilov in jih zaračuna partner, ki opravlja preglede.

Pri prodaji avtomobilov se prodajalcu zaračuna znesek v višini 1.8 % na ceno avtomobila. Ta del se izplača lastniku spletne aplikacije. Prav tako se zaračuna in izplača dinamično izračunan odstotek na sklenjena zavarovanja ter prevoze avtomobilov. Odstotek na sklenjena zavarovanja se zaračuna zavarovalnicam, odstotek na prevoze pa se zaračuna partnerjem, ki opravljajo prevoze avtomobilov.

Dodatne možnosti zaslužka, ki se lahko uporabijo v prihodnosti, so izpostavljeni oglasi avtomobilov ter tuji oglasi različnih avtomobilističnih storitev znotraj aplikacije. Možna je

tudi postavitev manjših specializiranih trgovin za večje trgovce z avtomobili. S tem bi za ceno mesečne rente večji trgovci pridobili svojo podstran znotraj aplikacije.

Pot do začetnih uporabnikov

Ker je pot do začetnih uporabnikov zelo težka, smo pripravili načrt, kako bomo začeli pridobivati uporabnike že v začetni fazi razvoja. Nekaj denarja bomo namenili oglaševanju preko Google AdWords in znotraj platforme Facebook, kjer bomo ciljali predvsem na avtomobilske navdušence. Nato jim bomo s pomočjo remarketinga (ki je opisan v nadaljevanju) ponovno posredovali oglase avtomobilov, ki so jih uporabniki že pregledovali. Na pomembnih avtomobilskih forumih in portalu Quora bomo predstavili aplikacijo v primernih forumskih temah. Že od začetka bomo merili promet ter izboljševali položaj v spletnih brskalnikih z optimizacijo SEO (Search Engine Optimization). Naročili bomo objave na pomembnejših avtomobilističnih spletnih straneh in začeli pisati svoj blog, ki bo dostopen v naši aplikaciji. S pomočjo bloga bomo zbirali elektronske naslove, na katerih bo zasnovan e-mail marketing. Blog bo vseboval pregled avtomobilističnih dogajanj, trendov in tudi pregled razvoja aplikacije Autocommerce.

Pot do začetne baze avtomobilov

Za začetno zapolnitev spletne strani smo uporabili že pregledane avtomobile naročnika aplikacije. Nekatera tuja podjetja že imajo interne baze preverjenih avtomobilov. Do teh baz lahko dostopajo samo prodajalci avtomobilov, ki imajo z njimi sklenjeno pogodbo. Večino avtomobilov iz internih baz se proda v roku enega tedna. V Nemčiji prodajo okoli 2500 takšnih avtomobilov na teden. Ko bo naročnik sklenil sporazum za dostop do obstoječih baz, pa bomo dodali še avtomobile iz teh baz.

Prednost pred konkurenco

Največja konkurenca naše spletne aplikacije je že obstoječa spletna aplikacija mobile.de, ki se ukvarja s prodajo rabljenih vozil in je največji ponudnik takšnih storitev v Evropi. Naš namen ni neposredno tekmovanje z mobile.de in s podobnimi spletnimi aplikacijami, ampak poskus spremembe in izboljšanje celotnega ekosistema za prodajo rabljenih vozil.

Prednost pred konkurenco želimo doseči z uvedbo pregleda vseh vozil, enostavnejšo in hitrejšo prodajo (prevoz in zavarovanje), popolno transparentnostjo ter dodatnim izločevanjem vmesnih členov med prodajalci in kupci. S tem bomo zagotovili nižje cene za kupce in hkrati višji dobiček za prodajalce rabljenih vozil.

2.3 Uporabljene tehnologije

V tem podpoglavju bodo na kratko opisane tehnologije, ki so bile uporabljene v okviru razvoja spletne aplikacije. Glavni namen je njihova predstavitev in utemeljitev ter pregled celotnega spletnega projekta. Dodali bomo tudi nekaj primerov njihove uporabe, bolj podroben razvoj celotne aplikacije pa bo predstavljen v naslednjem poglavju.

Kot temelj za prednji del (angl. front end) aplikacije smo na spletu izbrali odziven koncept uporabniškega vmesnika, imenovanega tema (angl. responsive admin dashboard template). V temi so že pripravljeni pogledi in koda, ki definira celoten izgled. Izgled aplikacije temelji predvsem na HTML, CSS in JavaScript kodi. Ime teme je **INSPINIA Theme** [1]. Temo smo vključili v Ruby on Rails projekt in jo priredili za potrebe aplikacije. Na začetku smo odstranili nepotrebne dele, kot so na primer že vnaprej pripravljene podstrani in izgledi, ki jih ne bomo uporabljali. Nato smo preuredili izgled posameznih podstrani in postavitev gradnikov. Tema vsebuje podobne gradnike, kot jih vsebuje ogrodje Bootstrap, le da imajo gradniki drugačen izgled in so bolj dovršeni. Vsebuje tudi nekaj JavaScript vtičnikov, ki skrbijo za uporabniške funkcije. Eden izmed njih je vtičnik za nalaganje datotek.

Uporaba tem za izdelavo aplikacij ni vedno primerna odločitev. Uporabljamo jih na projektih, kjer smo omejeni s časom in unikatni izgled aplikacije ni prioriteta. Takšen princip uporablja veliko zagonskih podjetij, saj s tem prihranijo veliko časa in se lahko posvečajo programiranju pomembnih delov aplikacije, ki tečejo v ozadju.

Aplikacija Autocommerce uporablja **Ruby on Rails (RoR)** ogrodje, ki deluje na **MVC** arhitekturi. Ogrodje RoR razširjajo paketi. Pomembnejši med njimi so Devise, Sqlite3, Refile [2], gmap4rails, Geocoder. Podrobneje jih bomo opisal v nadaljevanju skupaj z razvojem aplikacije. V aplikaciji je tudi paket ActiveAdmin, ki zelo enostavno razširja RoR aplikacijo in omogoča pregled nad podatki. Tako imamo pregled in kontrolo nad uporabniki, datotekami in vsemi naloženimi in uporabljenimi podatki v aplikaciji.

Na uporabniški strani uporabljamo **jQuery** knjižnico, ki razširja osnovno delovanje skriptnega jezika Javascript. Za prenose aplikacije na strežnik uporabljamo paket **Capistrano**, ki našo Ruby on Rails aplikacijo preslika iz repozitorija na GitHubu v repozitorij na strežniku.

Strežnik je bil usposobljen preko spletne storitve za gostovanje **Digital Ocean**, kjer uporabljamo privatni strežnik distribucije **Ubuntu 14.04**. Na njem teče Puma strežnik, ki je različica Nginx strežnika. S strežnikom komuniciramo preko **SSH** povezave v konzoli.

V aplikaciji uporabljamo tudi nekaj zunanjih storitev. Za analitiko uporabljamo storitvi **Google Analytic**s ter **Mixpanel**. Elektronska sporočila se pošiljajo preko storitve **SendGrid**, ekipa pa komunicira preko klepeta **Slack**.

2.3.1 Prednji del aplikacije (angl. front end)

Prednji del oziroma izgled naše aplikacije se nahaja v mapi /app/views in /app/assets. V prvi mapi se nahajajo izgledi posameznih podstrani. V drugi pa slikovne, JavaScript in CSS datoteke, ki so potrebne, da se te strani pravilno prikažejo.

V JavaScript in CSS datotekah na začetku definiramo tudi vtičnike in njihove uvozne poti. S tem dodamo vtičnike v projekt. Datoteke imenovane kot application.js in application.css so vrhovne datoteke, ki se prenesejo in izvajajo na vsaki strani aplikacije. Tukaj določimo stile in uvoze, ki so si enaki čez celotno aplikacijo.

Za osnovo v sprednjem delu uporabljamo ogrodje **Bootstrap** [3]**.** Bootstrap je velika zbirka CSS razredov in HTML značk, s katerim lahko zelo hitro in preprosto postavimo ogrodje in izgled spletne strani z uporabo Bootstrapovih gradnikov. Bootstrap sta začela prva razvijati Mark Otto in Jacob Thornton, ki sta bila zaposlena pri Twitterju. Ta ga je leta 2011 naredil odprtokodnega in ga ponudil za uporabo celotnemu spletu in od takrat je postal eden najbolj priljubljenih projektov na GitHubu.

Trenutna verzija je 3.3.6 in kmalu prihaja nova stabilna verzija 4.0. Bootstrap vsebuje veliko različnih komponent, kot so tabele, obrazci, gumbi in JavaScript komponente (npr. modalna okna, spustni meniji in ostale elemente). Prav tako pa je prilagodljivo ogrodje, ki prilagaja izgled strani za različne naprave. Danes je Bootstrap tako množično razširjen, da ga najdemo v veliki večini strani in predlog za spletne strani.

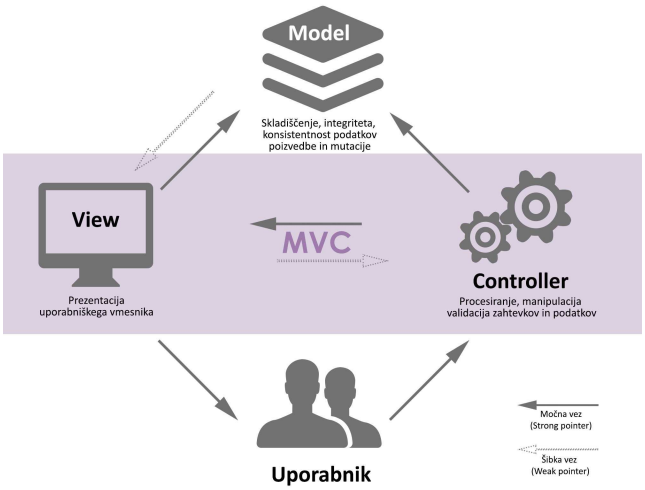
2.3.2 Zaledni del aplikacije (angl. back end)

Glavno ogrodje programske kode, s pomočjo katerega smo razvili aplikacijo, je ogrodje **Ruby on Rails** [4], ki postavlja strukturo in logiko celotnega projekta. RoR uporablja jezik Ruby. Paketi, ki razširjajo Ruby jezik, so dostopni preko Ruby paketov imenovanih Ruby Gems. Ogrodje RoR je eden izmed paketov.

Ruby on Rails je eno izmed najnovejših modernih ogrodij, ki se uporabljajo za hiter razvoj aplikacij in je zelo priljubljeno ogrodje zagonskih podjetij, ki želijo hitro razviti začetni produkt in ga stestirati na tržišču. Je ogrodje, ki deluje na Model-Pogled-Kontroler (MVC -

Model-View-Controler [5]) arhitekturi in omogoča vgrajeno strukturo s podatkovno bazo, logiko ter pogledi. S tem ločimo poslovno logiko in aplikacijske podatke od pogledov za uporabnika. Ogrodje sledi in vzpodbuja uporabo modernih konceptov razvijanja spletnih aplikacij, kot so 'konvencije pred konfiguracijami', 'ne ponavljaj se' in 'active record vzorec' (slika 3).

Poleg tega vsebuje generatorje, ki se uporabljajo za generiranje aplikacij oziroma posameznih delov aplikacije. S njihovo pomočjo lahko zelo hitro postavimo osnovno strukturo aplikacije. Ogrodje je bilo razvito leta 2004 in dobilo odprtokodni status leta 2005. Razcvet je doživelo leta 2007, ko ga je podjetje Apple vključilo v operacijski sistem Mac OSX. Trenutno je zadnja stabilna verzija ogrodja Ruby on Rails 5.0 RC 1, ki je tudi najnovejša pomembnejša izboljšava zaradi prehoda iz verzije 4.x na verzijo 5.x.



Slika 3: Predstavitev delovanja MVC arhitekture [6]

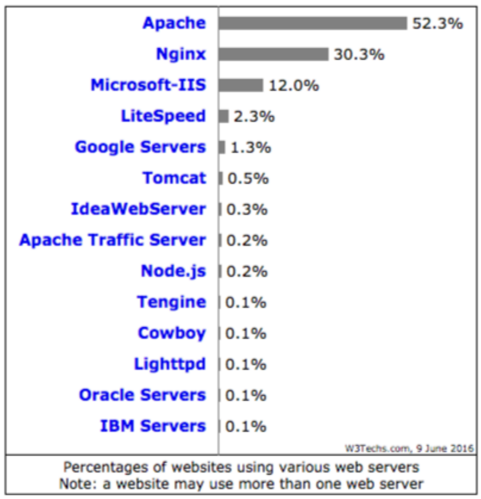
Glavni Ruby paketi v našem projektu so **Devise**, **Sqlite3**, **Refile**, **gmap4rails** in **Geocoder**. **Devise** je paket, ki definira uporabnika ter funkcije za avtentikacijo ter postavi vse poglede v aplikaciji, ki so namenjeni registraciji in prijavi uporabnika. Je eden izmed najbolj razširjenih paketov in se uporablja skoraj v vsakem RoR projektu. **Sqlite3** je podatkovna baza Sqlite, ki se privzeto uporablja v lokalnih RoR projektih. **Refile** je eden izmed novejših sistemov za prenašanje datotek na strežnik, ki uporablja sveže pristope, ki rešuje nekatere težave z zastarelimi paketi in vtičniki, ki so se močno uporabljali v preteklosti. Refile je še vedno v beta fazi. V našem primeru se uporablja za prenašanje in urejanje slik vozil. **Gmap4rails** je paket, ki nam omogoča delo z Google mapami. Uporabljamo jih za prikaz najkrajše razdalje med kupcem in vozilom. **Geocoder** opravlja izračune med različnimi lokacijami, ki predstavljajo razdalje med kupcem in vozilom.

2.3.3 Strežnik

Spletni strežnik je računalniški sistem, ki je zmožen procesirati HTTP zahteve in na njih odgovarjati. Z izrazom spletni strežnik lahko označimo celoten sistem (računalnik in programsko opremo) ali pa samo programsko opremo.

V našem projektu uporabljamo spletni strežnik **Puma**, ki je različica **Nginx** strežnika. Nastavitvene datoteke se v RoR projektu nahajajo v mapi /config ter v datotekah **nginx.conf** in **deploy.rb**. Prva datoteka konfigurira Nginx nastavitve, druga pa hrani ukaze in procedure, ki se zgodijo ob nalaganju aplikacije na strežnik.

Apache, Nginx in Microsoft-IIS so spletni strežniki (programska oprema), ki skupaj tvorijo več kot 94.6 % delež vseh spletnih strežnikov, zato se z njimi sreča vsak spletni razvijalec (slika 4).



Slika 4: Delež programske opreme za strežnike [7]

Spletni strežnik **Apache** [8], velikokrat samo Apache, je spletni strežnik, ki igra ključno vlogo pri širjenju spleta. Bil je prva alternativa Netscapeovemu spletnemu strežniku, trenutno znanemu kot spletni strežnik Sun Java System. Od aprila 1996 je Apache najbolj popularen HTTP strežnik na celem spletu. Od oktobra 2007 pa je bilo na Apachijevem strežniku postavljenih približno 48 % vseh spletnih strani. **Nginx** [9] je alternativa popularnemu spletnemu strežniku Apache. Je odprto kodni reverzni proxy za HTTP, HTTPS, SMTP, POP3 in IMAP protokole. Uporablja se lahko tudi kot strežnik za porazdelitev obremenitve (ang. load balancer), HTTP predpomnilnik in spletni strežnik (razvit kot spletni strežnik). Servira lahko statično vsebino, preko FastCGI, SCGI, uwsgi ali memcached protokolov pa lahko tudi dinamično.

Po študiji Netcraft-a, izvedeni januarja 2015, je Nginx zadolžen za serviranje 14.61 % vseh spletnih strani. Strežnik je znan po svoji hitrosti, nizki porabi sistemskih sredstev ter zmožnostjo dela z veliko hkratnimi povezavami. Izdan je pod Free BSD licenco ter deluje na platformah in operacijskih sistemih, kot so FreeBSD, Linux, Solaris, AIX, HP-UX, Mac OS X ter Windows. Nginx poganja spletne strani, kot so Netflix, Hulu, Pinterest, CloudFlare, Airbnb, WordPress.com, GitHub, SoundCloud, Zynga, Eventbrite, Zappos, Media Temple, Heroku, RightScale, Engine Yard in MaxCDN.

Operacijski sistem za strežnik

Na računalniku, ki deluje kot strežnik, teče operacijski sistem. Operacijski sistem je program, ki ga računalnik uporablja za najbolj osnovno delovanje. Strežnik lahko deluje na eni izmed distribucij Unixa ali na Windows Serverju. Unix strežniki predstavljajo 67.5 % delež, Windows Server strežniki pa 32.4 % delež. Ostali delež je zanemarljiv in pripada OS X-u. Na našem strežniku uporabljamo operacijski sistem **Ubuntu**, ki je različica Unixa, oziroma bolj natančno Linuxa.

2.3.4 Podatkovna baza

PostgreSQL

PostgreSQL podatkovno bazo uporabljamo na strežniku, kjer teče aplikacija. PostgreSQL razširja delovanje lokalnega podatkovne base Sqlite, zato je bolj primerna, da jo uporabimo v produkciji. V aplikacijo ga enostavno naložimo z pripisom paketa v Gemfile, ki jo nato naložimo in nastavimo. Konfiguracija poveže podatkovno bazo z aplikacijo in omogoča medsebojno komunikacijo.

Glavne značilnosti PostgreSQL-a so: podpora na vseh platformah, objektno - relacijska podatkovna baza, odprtokoden produkt, referenčna integriteta, razširljivost, PITR (point in time recovery).

Primer vključitve PostgreSQL paketa v RoR projekt:

gem 'pg'

Po definiciji je PostgreSQL objektno relacijska podatkovna baza z dolgo zgodovino. Njeni začetki segajo v leto 1970, ko so na kalifornijski univerzi Berkeley pričeli z razvojem podatkovne baze pod imenom Ingres. Relacijska tehnologija je Ingres kmalu obrnila v komercialne vode, produkt je prevzelo podjetje Computer Associates. Okoli leta 1986 je

Michael Stonebraker na univerzi Berkeley vodil ekipo, ki je relacijski podatkovni bazi »vsadila« objektno jedro. Novo, objektno relacijsko bazo, so poimenovali Postgres, ki pa je bila ponovno komercializirana, kupilo jo je podjetje Illustra (del Informix Corporation).

V sredini 90. let sta Andrew Yu and Jolly Chen dodala bazi SQL podporo. Prejšnja verzija je namreč uporabljala specifičen poizvedovalni jezik – postquel1 . Leta 1996 so dodali še nekaj dodatnih funkcionalnosti in izboljšav (MVCC transakcijski model, polna podpora jezika SQL 92). Dobila je tudi novo ime, pod katerim jo poznamo danes, to je PostgreSQL [10].

SQLite

Omenjeno podatkovno bazo privzeto uporabljamo v lokalnem projektu RoR. V produkciji jo velikokrat nadgradimo z uporabo PostgreSQL. SQLite je relacijski podatkovni sistem, ki je bil razvit v programskem jeziku C. Za razliko od ostalih podatkovnih sistemov SQLite ne deluje na principu strežnik-odjemalec, ampak je vgrajen neposredno v aplikacijo oziroma program. Razvit je bil leta 2000 in je odprtokodni produkt, ki je zaradi razširjenosti redno posodobljen.

Ker je sistem izredno majhen, ampak vseeno zelo zmogljiv, se uporablja predvsem v manjših aplikacijah, ki ne opravljajo zahtevnih operacij na veliki količini podatkov. Zato je izjemno primeren za aplikacije, ki na začetku potrebujejo hiter razvoj in dovolj fleksibilnosti.

Privzeto se uporablja na večini spletnih ogrodij, kot so Ruby on Rails, Django, Bugzilla in web2py. Prav tako se uporablja tudi v OpenBSD, Androidu, Smybian OSu ter Windows 10.

2.4 Ostala orodja

V naslednjem podpoglavju bodo opisana pomembna orodja in zunanje storitve, ki nam olajšajo delo z analitiko, komunikacijo in pridobivanje novih uporabnikov. Razvijanje storitev ni le razvoj aplikacije, ampak zagotavljanje nemotenega delovanja širšega sistema, ki obsega prav vse dele aplikacije. Obsega tudi vse ljudi, ki so vpleteni v razvoj. Ker moramo poleg razvoja skrbeti še za mnogo drugih aspektov (uporabniško podporo, nemoteno delovanje elektronskih sporočil, analitiko, marketing, socialna omrežja, komunikacijo in produktivnost v razvijalski ekipi, ipd.), se poslužujemo integracije z ostalimi spletnimi storitvami. To ponavadi storimo z že vnaprej pripravljeno integracijo, API vmesnikom ali z dodajanjem paketa k našemu projektu. Storitve in orodja so razvrščene v smiselne kategorije.

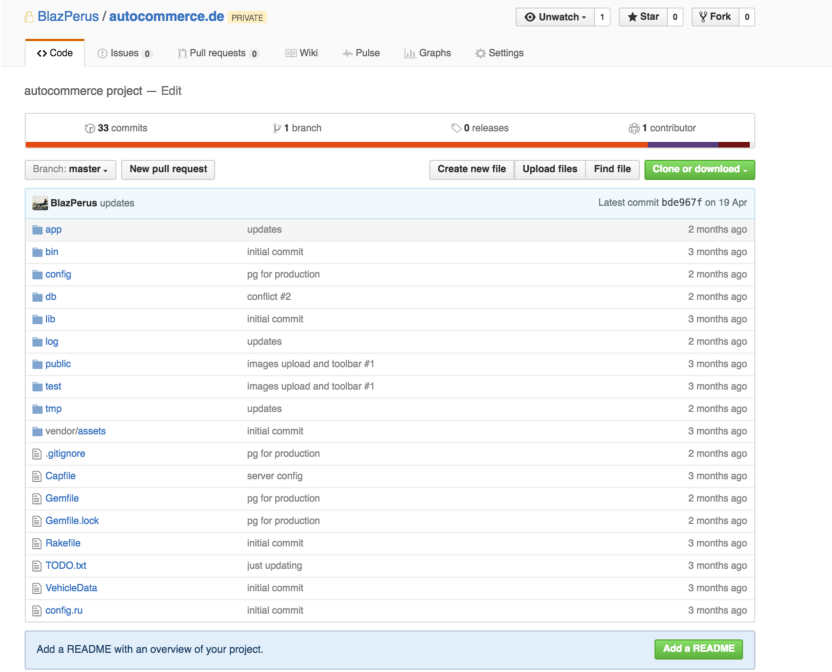
Storitve elektronske pošte (e-mail)

Pošiljanje elektronskih sporočil je ena izmed osnovnih funkcij, ki jih opravlja spletna aplikacija. Za pošiljanje sporočil lahko uporabimo SMTP strežnik, ki ga konfiguriramo znotraj naše aplikacije ali pa uporabimo eno izmed bolj zanesljivih zunanjih storitev, ki je specializirana za pošiljanje elektronskih sporočil. Takšni storitvi sta na primer MailGun ali SendGrid. Maja leta 2016 je MailGun postal privzeto plačljiva storitev, zato se je veliko aplikacij odločilo za prehod na SendGrid, ki je plačljiv le v primeru, da pošljemo na mesec več kot deset tisoč elektronskih sporočil. Storitvi sta si med seboj zelo podobni. Vsaka ponuja dostop do API in možnost vodenja analitike ter najrazličnejše naprednejše funkcije. Večina aplikacij uporablja zunanje storitve za pošiljanje elektronskih sporočil, ker so bolj zanesljive in ponujajo večji nabor funkcij kot lastni strežniki. V naši aplikaciji uporabljamo storitev **SendGrid** [11], ki smo jo konfigurirali s pomočjo konfiguracijske datoteke **setup_mail.rb**, ki se nahaja v mapi **/config**. Koda za integracijo je sledeča:

```
ActionMailer::Base.smtp_settings = {
   address:          "smtp.sendgrid.net",
   port:             587,
   domain:           "autocommerce.com",
   user_name:        “naše_uporabniško_ime”,
   password:         “naše_geslo",
   authentication:   "plain",
   enable_starttls_auto: true
}
ActionMailer::Base.default_url_options[:host] = “mail.autocommerce.com”
```

Shranjevanje kode in kontrola verzij

V procesu razvoja spletnih aplikacij je potrebno sprotno shranjevanje programske kode in sinhronizacija med programerji. Za takšne namene se uporabljajo sistemi oziroma programi, ki skrbijo za nadzor verzij. V praksi to pomeni, da programerji uporabljajo program, ki skrbi za nadzor, in oddajajo programsko kodo v računalniško mapo, ki se nahaja na oddaljenem strežniku. Strežnik skrbi za sinhronizacijo med programerji in tako poskrbi, da ne pride do kršitev zaporedja in s tem vzdržuje pravilno časovnico projekta.



Slika 5: Primer projektne podstrani na spletni strani github.com

Za spletno aplikacijo Autocommerce smo uporabili shranjevalnik **Git**. Zadnja leta je postal najbolj priljubljen in zanj obstaja tudi priljubljena platforma **GitHub**. Ta ponuja uporabnikom bolj prijazen grafični vmesnik in zelo zmogljivo spletno stran, ki še dodatno olajša nadzor nad programsko kodo in verzijami (slika 5).

Trenutno ima GitHub močan monopol nad programsko opremo, ki skrbi za nadzor nad verzijami, in je ena izmed najbolj priljubljenih spletnih strani. Ima več kot 14 milijonov uporabnikov in več kot 450 zaposlenih. Git je razvil Linus Torvalds leta 2005. Orodje Git je na začetku imenoval kot “neumno orodje za nadzor nad vsebino”. Ime je utemeljil kot naključno zaporedje treh črk, ki se lahko enostavno izgovarja in še ni uporabljeno kot UNIX ukaz.

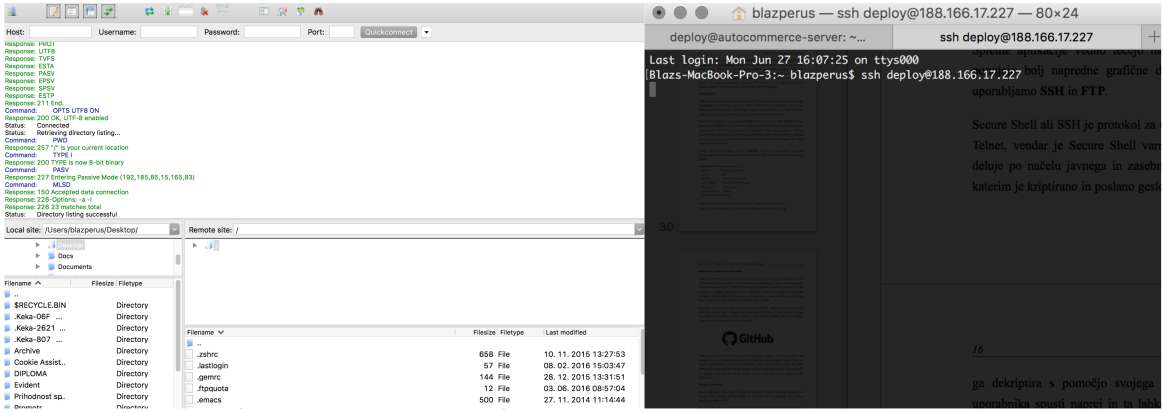
Dostop do strežnika

Spletne aplikacije vedno tečejo na spletnih strežnikih. Nekateri ponudniki gostovanj že ponujajo bolj napredne grafične dostope do strežnikov. Mi za dostopanje do strežnika uporabljamo **SSH** [12] in **FTP** [13] (slika 6).

Secure Shell ali SSH je protokol za upravljanje računalnika na daljavo. Isto funkcijo ima tudi Telnet, vendar je Secure Shell varnejši, saj pošlje odjemalcu geslo v kriptirani obliki. SSH deluje po načelu javnega in zasebnega ključa. Strežniku najprej pošlje javni ključ, s katerim je kriptirano in poslano geslo SSH.

Takoj, ko SSH-strežnik prejme kriptirano geslo, ga dekriptira s pomočjo svojega zasebnega ključa. Če se gesli ujemata, SSH-strežnik uporabnika spusti naprej in ta lahko prične z delom na računalniku (po pregledu gesel še dodatno preveri, če ima uporabnik pravico uporabljati konzolo). Za uporabo SSH-ja so potrebni posebni odjemalci.

Spodaj je primer povezave na naš spletni strežnik preko SSH iz konzole. FTP je protokol za prenos datotek (File Transfer Protocol) in se uporablja za prenašanje in upravljanje z



Slika 6: Primer FTP dostopa z odjemalcem Filezilla (levo) in SSH v konzoli (desno)

datotekami na strežniku. Mi uporabljamo FTP odjemalec Filezilla, ki olajša povezovanje na strežnik. V odjemalcu vpišemo naslov, uporabniško ime in geslo.

Nato se prične povezovanje na strežnik, ki nam ob uspešnem vpisu izpiše imena datotek na strežniku. Potem smo pripravljeni za delo z datotekami na strežniku.

Paketni menedžerji

Paketni menedžerji so sistemi za standardizirano distribucijo delčkov programske kode (paketov). Vsak izmed popularnejših programskih jezikov in platform (na primer OS X ali Android) uporablja paketni menedžer. Paketi so manjši programčki in knjižice, ki opravljajo določeno funkcijo. Ker se v različnih spletnih aplikacijah posamezni deli aplikacije ponavljajo (kot na primer avtentikacija uporabnikov), se takšne dele zapakira v pakete. Tako lahko razvijalci prenesejo specifičen paket in ga hitro vključijo v projekt ter uporabijo v svoji aplikaciji.

Ruby uporablja RubyGems, na kratko imenovane Gems (slovensko 'dragulji'). Paketni menedžer za Ruby on Rails je **Bundler**, ki nadzoruje pakete v RoR projektu. Odpravlja medsebojne napake in posodablja pakete, ko je to potrebno. Bundler uporablja datoteko Gemfile, kjer so zapisani vsi paketi, ki jih trenutni RoR projekt uporablja. Ko v konzoli zaženemo ukaz **bundle**, se prične nalaganje novih paketov in odstranjevanje zastarelih. Pakete lahko posodobimo z uporabo ukaza **bundle update**. Primer ukaza bundle je prikazan na sliki 7, ko se ob klicu ukaza izpišejo vsi že vgrajeni paketi. Z zeleno barvo pa se obarvajo na novo naloženi paketi.

Slika 7: Primer ukaza bundle

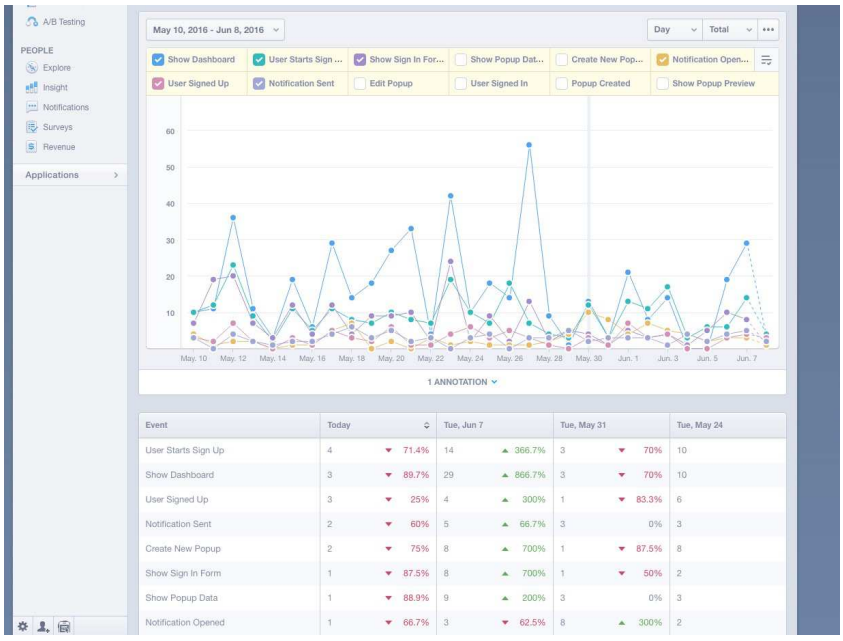
Analitika

Analitika je eno izmed najmočnejših orodij na podlagi katerih lahko gradimo spletne storitve. S pomočjo analitike vodimo evidenco obiska, predvidevamo trende ter dobivamo povratne

informacije od uporabnikov. Z njeno pomočjo lahko spreminjamo delčke aplikacije, ki se ne obnašajo, kot bi se morali, ter izboljšujemo razvoj aplikacije in marketinga. Najbolj uporabljena orodja za analitiko so Google Analytics (GA), Mixpanel, KISSmetrics, CrazyEgg, ki jih uporabljamo tudi mi.

Google Analytics uporabljamo za osnovni pregled obiska naše spletne aplikacije. Pri tem nas zanima predvsem mesečni obisk, odstotek obiska samo ene strani ter število vračajočih se uporabnikov. Zanima nas tudi, od kod prihajajo uporabniki ter od kod so prišli do nas. Googlova analitika nam služi tudi kot dokaz obiska pri investitorjih ter oglasnih ponudbah. V spletno stran je vgrajena z delčkom JavaScript kode, ki se prenese ob odpiranju strani. S pomočjo GA serviramo tudi boljše oglase s pomočjo oglasnega sistema Ad Words.

Mixpanel je storitev, ki nudi bolj specifično analitiko za vsakega uporabnika. Takšna analitika zahteva več premisleka in programerskega časa, čeprav ni zelo zahtevna za integracijo. Z njeno pomočjo lahko spremljamo vsak premik uporabnika ter tako dobivamo bolj natančne informacije o delovanju. Spremljamo lahko premike po aplikaciji, klike, dogodke ter število nakupov (slika 8).



Slika 8: Prikaz analitike v storitvi Mixpanel

Storitev **KissMetrics** uporabljamo predvsem zaradi tega, ker nam omogoča izpis mape, ki nam pokaže kateri deli aplikacije so najbolj uporabljeni. S takšnimi informacijami lahko bolje zasnujemo začetne strani in podstrani aplikacije.

Marketing

Marketing spletnih storitev je zelo širok pojem, zato bomo zaobjeli samo osnovnejše principe in orodja, ki jih danes uporabljajo zagonska podjetja za širitev in rast. Najpomembnejše tri stvari, ki jih želimo doseči z uporabo različnih marketinških tehnik, so povečanje števila uporabnikov, povečanje prodaje vozil in razvoj prepoznavnosti blagovne znamke.

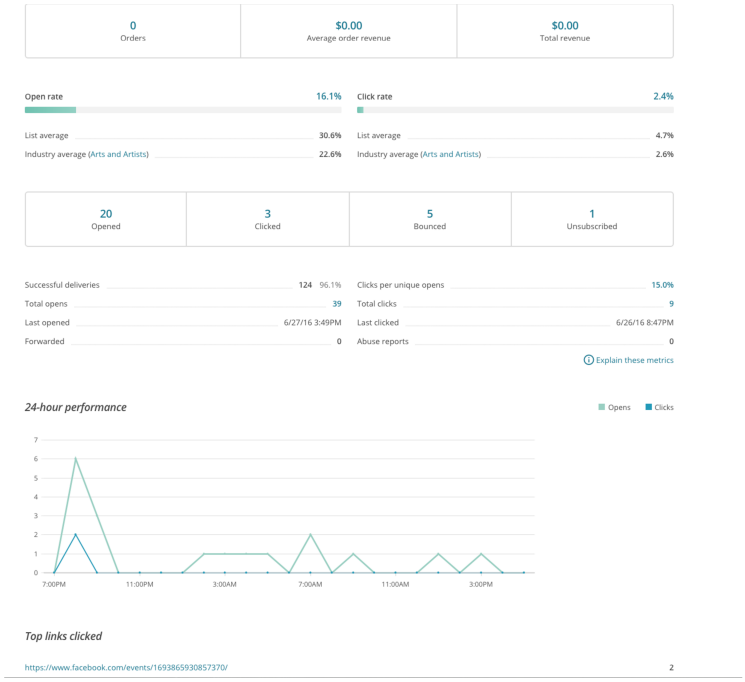
Ker imajo mlada podjetja zelo malo kapitala, ki bi ga lahko usmerile v plačan marketing, se poslužujejo tehnik za povečanje obiska in prepoznavnosti za katere namenijo zelo malo denarja. Takšno obliko marketinga danes imenujemo "**growth hacking**" in jo uporabljamo tudi mi. Ustreznega slovenskega prevoda za zdaj ni – je nekakšna mešanica oziroma hibrid principov, ki zaobjema in prepleta kodiranje, analitiko in spletni marketing.

Ena izmed osnovnejših oblik marketinga je prisotnost na **socialnih omrežjih.** S svojo prisotnostjo osveščamo ljudi o našem podjetju oziroma storitvi ali produktu. Socialna omrežja kot npr. Facebook imajo izvrstno razvit sistem oglaševalskih orodij, s katerimi zelo natančno dosežemo ciljne skupine. Zelo primerna omrežja so tudi Twitter, Instagram, Snapchat, LinkedIn in Quora. Ker se vsako omrežje uporablja malenkost drugače ter uporablja drugačne principe komunikacije, je zelo pomembno, da za vsako platformo prilagodimo način komuniciranja.

Google nudi nekaj visokotehnoloških rešitev za marketing in analitiko. Ena izmed njih je Google Analytics, ki je najbolj razširjena analitična storitev na spletu. S pomočjo GA lahko povežemo tudi storitev **AdWords**, ki je Googlova platforma za oglaševanje. Pomembne so tudi SEO (angl. Search Engine Optimization) tehnike, ki jih uporabimo na naših spletnih aplikacijah. S pomočjo teh tehnik omogočimo iskalniku Google, da naše spletne aplikacije na strani rezultatov iskanj postavi višje. Ker je vsak premik na iskalnih pozicijah pomemben, se uporabljajo orodja, kot so **RankTrackr** s katerimi sledimo položaj naših spletnih strani za določene besedne zveze.

V povezavi z oglasnimi sistemi kot je AdWords uporabljamo zunanja orodja za spletni remarketing. Remarketing deluje tako, da uporabniku nastavimo spletni piškot, ko obišče našo spletno stran. Nato mu preko oglasnih sistemov serviramo oglase na tujih spletnih straneh in socialnih omrežjih. Remarketing deluje tudi znotraj sistema AdWords in ne potrebujemo zunanjih orodij. Prednost le teh pa je v tem, da se takšna orodja specializirajo samo za remarketing in so zato bolj dovršena na tem področju. Nekatera izmed takšnih orodij so AdRoll, Facebook Email Remarketing in Hubspot.

Zelo pomembno področje je tudi **e-mail marketing**. Tukaj se uporabljajo zunanje storitve, kot so Infusionsoft, MailChimp, iContact. Omogočajo naprednejše email funkcije, kot so avtomatična sporočila glede na dogodke, ki se zgodijo posameznim uporabnikom. Tako dobi vsak uporabnik prilagojena sporočila glede na to, kako uporablja aplikacijo. S tem avtomatiziramo nekatere naloge, katerih namen je predvsem povečanje interakcije z uporabnikom, ki ga želimo prepričati v nakup našega izdelka ali storitve. V aplikaciji Autocommerce še ne uporabljamo naštetih storitev, ker še nismo na stopnji, ko bi bilo to pomembno. Zaenkrat se uporablja samo MailChimp za občasno pošiljanje elektronskih obvestil naročenim uporabnikom. Uporabniki se lahko na novice naročijo na začetni strani aplikacije, kjer jih pričaka obvestilo za vpis. MailChimp omogoča vodenje baze z elektronskimi naslovi ter analitiko za elektronska sporočila. To nam omogoča, da spremljamo uporabnike, ki so sporočila prebrali, in vidimo, katere povezave v elektronskem sporočilu so odprli. Slika 9 prikazuje primer takšnega analitičnega poročila za poslano e-mail kampanjo.



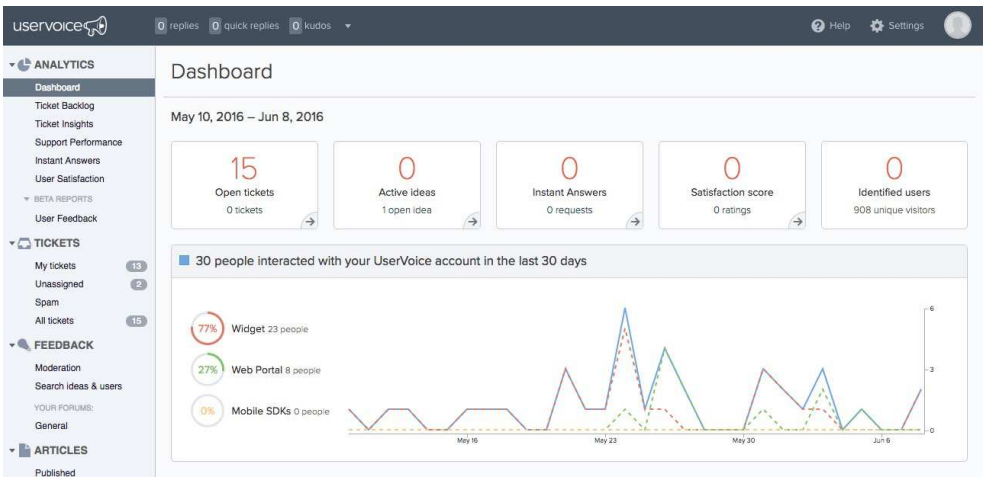
Slika 9: Primer analitičnega poročila v MailChimpu

Push notifications (potisna sporočila) so krajša sporočila, ki obveščajo uporabnike o dogajanju v aplikaciji. Sporočila se pošiljajo na vse platforme in naprave, ki jih integriramo v aplikacijo. Na splošno se ločijo na sporočila za prenosne naprave in sporočila za računalnike. Njihov namen je povečanje interakcije z aplikacijo. So bolj moteča, zato se za njih odloči manjše število uporabnikov.

Njihova prednost je v tem, da so zelo učinkovita in fleksibilna za uporabo. Sporočila so dostavljena do uporabnikov tudi, ko uporabljajo druge aplikacije. Mi uporabljamo storitev OneSignal, ki omogoča lahko integracijo, pregled nad naročenimi uporabniki in pošiljanje takšnih sporočil.

Predstavitvena stran produkta, storitve ali podjetja je danes nepogrešljiva oblika marketinga. Deluje kot baza in temelj veliko marketinških tehnik za povečanje prodaje. Namen teh strani je izključno v tem, da predstavijo produkt, storitev ali podjetje in napeljejo uporabnika k nakupu ali k registraciji.

Uporabniške podpore večina ljudi ne bi pristela k marketingu, čeprav je eden izmed najpomembnejših delov spletnega marketinga. Uporabniki si velikokrat zaželijo človeške podpore, zato je zelo pomembno, da imajo dobro izkušnjo pri uporabniški podpori. Predvsem je pomembno, da je podpora hitra in učinkovito odgovarja ter razrešuje težave, ki nastanejo pri uporabi aplikacij. Za aplikacijo uporabljamo storitev UserVoice, ki skrbi za vodenje uporabniške podpore (slika 10).



Slika 10: Primer pogleda uporabniške podpore v storitvi UserVoice

Sistemi za hitro posredovanje vsebine (CDN)

CDN (Content Delivery Network) sistemi so sistemi, ki uporabnikom aplikacij dostavljajo vsebino z več lokacij na svetu. To so sistemi, ki delujejo kot gruče strežnikov, katerih namen ni procesiranje zahtev, ampak hitra dostava podatkov. Uporabljajo se tudi za hitro dostavljanje statičnih spletnih vsebin, kot so prevedene CSS in HTML datoteke.

Ker so takšni sistemi kompleksni, se za njih odločajo predvsem podjetja, ki že imajo visoko razvite spletne storitve z veliko uporabniki. V zadnjem času so se pojavili tudi bolj enostavni

sistemi, ki jih lahko uporabljajo manjše spletne storitve in ne potrebujejo kompleksnih integracij. Eden izmed popularnejših je Cloudfront, ki ga je razvil Amazon.

Naša aplikacija takšnih sistemov še ne uporablja. V primeru zelo visoke rasti obiska v aplikaciji na različnih kontinentih bomo primorani v integracijo takšnih sistemov.

Varnost

Zagotavljanje varnosti je visoka prioriteta vseh razvijalcev, saj večina aplikacij hrani občutljive informacije. Varnost se začne pri pravilni konfiguraciji spletnih strežnikov ter sledenju programerskih konvencij, s katerim se izognemo vdorom.

Ena izmed najbolj osnovnih stvari, ki jih lahko programer stori, je da na svoj spletni strežnik naloži SSL certifikat [14] (slika 11). **SSL** (Secure Socket Layer) certifikat je digitalno varnostno potrdilo, ki zagotavlja pristnost identitete spletnega mesta, hkrati pa šifrira vse podatke, ki se prenašajo med spletno stranjo in njenimi obiskovalci. SSL certifikat torej poskrbi za varnost prenosa podatkov. Takšne certifikate izdajajo spletni ponudniki gostovanj ter specializirane certifikatne agencije. Njihove cene se začnejo pri 10 evrih na leto in segajo vse do dva tisoč evrov na leto.

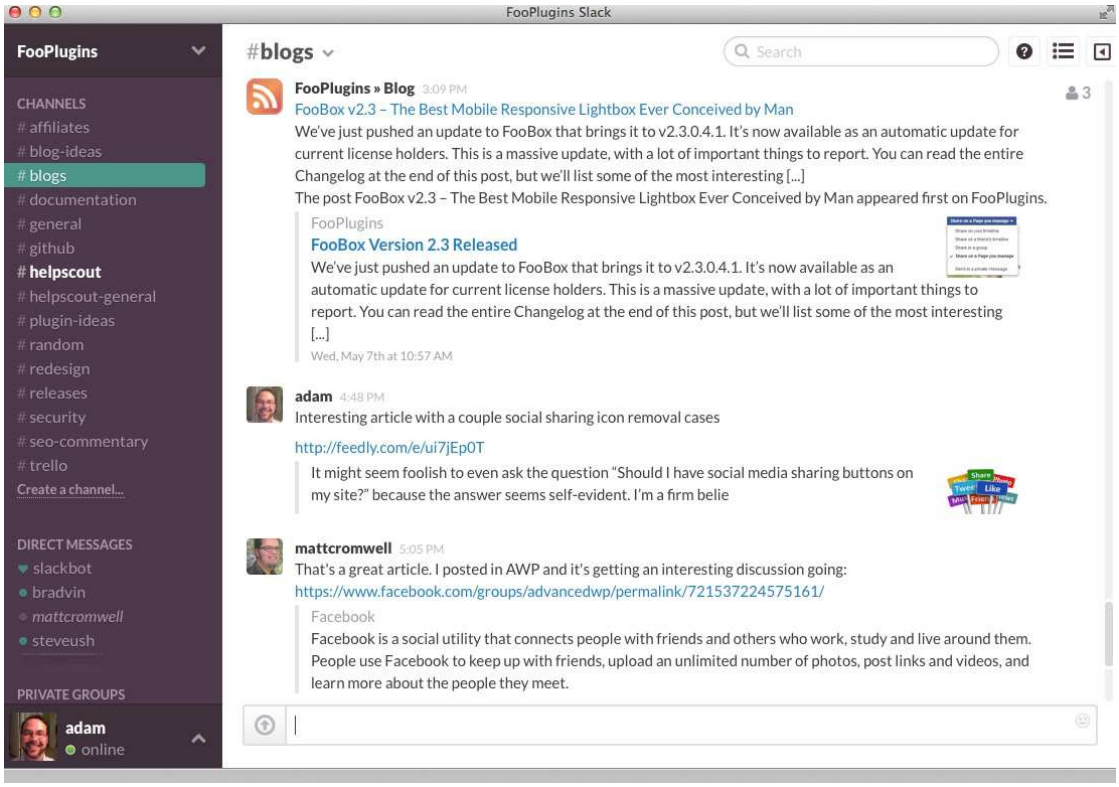
Slika 11: Primer povezave, ko spletna stran deluje preko SSL

Varnost na strežniku smo vzpostavili tako, da smo občutljivim konfiguracijskim datotekam omejili dostop. Večina datotek ima pravice, ki jih s številko zapišemo kot 644 pravice. S tem omejimo dostop neavtoriziranim uporabnikom in močno povečamo varnost. Na strežniku uporabljamo SSL certifikat, kar zagotavlja šifrirano povezavo z strežnikom. Tako je povezava med prenašanjem občutljivih podatkov varna. Na primer ob nakupih in vnosih gesel. SSL se privzeto uporablja čez celotno aplikacijo.

Komunikacija

Komunikacija je v ekipah razvijalcev pomemben vidik, ki ga je potrebno upoštevati pri razvoju aplikacij in podjetij. Boljša kot je komunikacija, hitreje teče razvoj projektov. Danes se pri večini zagonskih podjetij uporablja za dnevno komuniciranje storitev **Slack** [15] (slika 12). Je orodje za komuniciranje v obliki spletnega klepeta in deluje kot klepet za sodelovanje vseh vrst ekip.

Slack je bil izdan leta 2013 in je v nekaj letih postal impresivno orodje, s pomočjo katerega komunicirajo ekipe po vsem svetu. Omogoča veliko različnih integracij med spletnimi storitvami kot so Google Drive, Trello, Dropbox, GitHub in podobnimi.



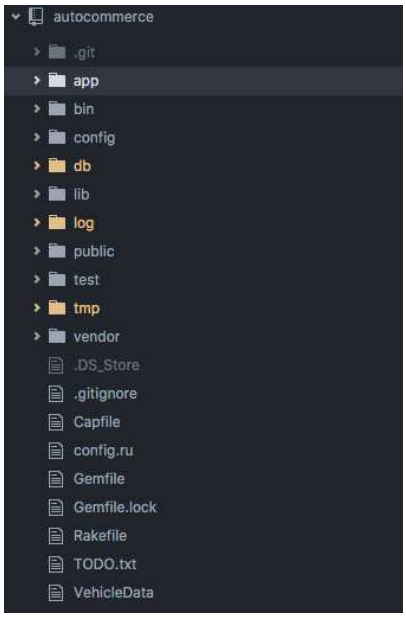
Slika 12: Primer klepeta v Slacku

3. Razvoj aplikacije

V okviru diplomske naloge smo razvili aplikacijo **Autocommerce**. V tem delu bomo opisali posamezne dele aplikacije ter podrobneje opredelili, čemu služijo, ter dodali delčke programske kode. Kot že omenjeno že v prejšnjem poglavju, je bila naša aplikacija razvita s pomočjo **Ruby on Rails** ogrodja.

3.1 Osnova in zaledni del

Na začetku smo kreirali novo RoR aplikacijo, kar nam je ustvarilo prazen projekt z vsemi privzetimi datotečnimi mapami in datotekami. Vsak RoR projekt je razdeljen v več podmap, ki sestavljajo strukturo projekta. V vsaki podmapi so datoteke, ki so namenjene specifičnemu delu aplikacije. Prikaz strukture projekta je viden na sliki 13. Večji del aplikacije je v podmapi **app**, kjer se nahajajo kontrolerji, modeli in pogledi. Poleg tega se tukaj nahajajo tudi vse JavaScript in CSS datoteke, krajši pomočniki ter ostali delčki kode. V teh mapah se v procesu razvijanja nahajamo večino časa. V pogledih se nahaja koda, ki sestavlja izgled strani. V kontrolerjih je logika aplikacije, ki se izvaja na podlagi vhodov in stanj. V modelih pa je logika ravnanja s podatkovno bazo.



Slika 13: Struktura Ruby on Rails projekta

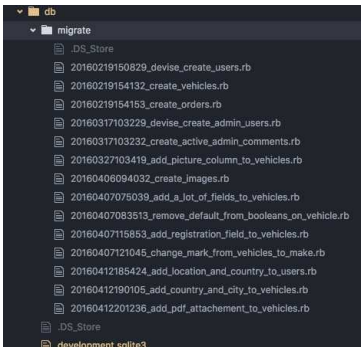
V podmapi **db** se nahajajo datoteke, ki so potrebne za delovanje podatkovne baze. V podmapi **config** pa vse datoteke za konfiguracijo. Tukaj se nahajajo poleg vseh ostalih datotek za konfiguracijo tudi datoteke, s katerimi konfiguriramo celoten projekt, pakete, podatkovno bazo in urejamo aplikacijske poti. Za našo aplikacijo smo uredili datoteke tako, da smo dodali aplikacijske poti do podstrani v aplikaciji v datoteko za poti (**routes.rb**). Dodali smo gesla za podatkovno bazo (**database.yml**), ki jo uporabljamo na produkcijskem strežniku ter nastavitve za strežnik (Puma Nginx).

Podatkovna baza

Privzeto se na lokalnih projektih v RoR uporablja SQLite podatkovna baza. Na strežniku pa uporabljamo PostgreSQL, ki omogoča dodatne funkcije poizvedb in je bolj primerna za aplikacije v produkciji. Konfigurirali smo jo tako, da smo se z uporabo SSH povezali na strežnik. Na strežniku smo namestili PostgreSQL in ustvarili novega uporabnika, ki ima dovoljenje za dostop do podatkovne baze. Na koncu smo bazo povezali z našo aplikacijo, tako da smo uredili konfiguracijske datoteke in vnesli podatke za dostop in geslo. Dostopni podatki se nahajajo v datoteki **/config/database.yml**, ki se jo izključi iz repozitorija na GitHubu in ročno prenese na produkcijski strežnik. S tem se izognemo temu, da bi shranjevali gesla na GitHubu in povečamo varnost aplikacije.

Shema podatkovne baze in podatkovne migracije

Shema podatkovne baze zaobjema vse tabele in entitete ter entitetne tipe, ki jih vsebuje podatkovna baza aplikacije. Dostopna je v datoteki **schema.rb**, ki se nagaja v mapi **/db**. S pomočjo migracij lahko enostavno urejamo sestavo podatkovne baze. Migracije uporabljajo koncept svežnja in se med njimi pomikamo nazaj ali naprej. Trenutni projekt vsebuje štirinajst migracij, za vsako migracijo se ustvari nova datoteka (slika 14).



Slika 14: Podatkovne migracije projekta

Predstavljenih je štirinajst migracij, ki so bile ustvarjene v času razvoja. Začetno migracijo je pognal paket Devise, ki je ustvaril uporabnika. Naslednjo smo ustvarili ročno. V tej migraciji smo dodali podatkovna polja v podatkovno bazo, ki jih bomo uporabljali za prikaz podatkov za vozila. Naslednje migracije so podobne. Dodajali smo podatkovna polja in ustvarjali nove podatkovne modele.

Spodaj je programska koda ene izmed podatkovnih migracij. V njej smo dodali podatke za lokacijo k podatkovnemu modelu uporabnika. Obe entiteti sta tekstovnega tipa, ki ne smeta biti prazni in imata privzeti vrednosti.

```
class AddLocationAndCountryToUsers < ActiveRecord::Migration
  def change
    add_column :users, :city, :string, null: false, default: "Berlin"
    add_column :users, :country, :string, null: false, default: "Germany"
  end
end
```

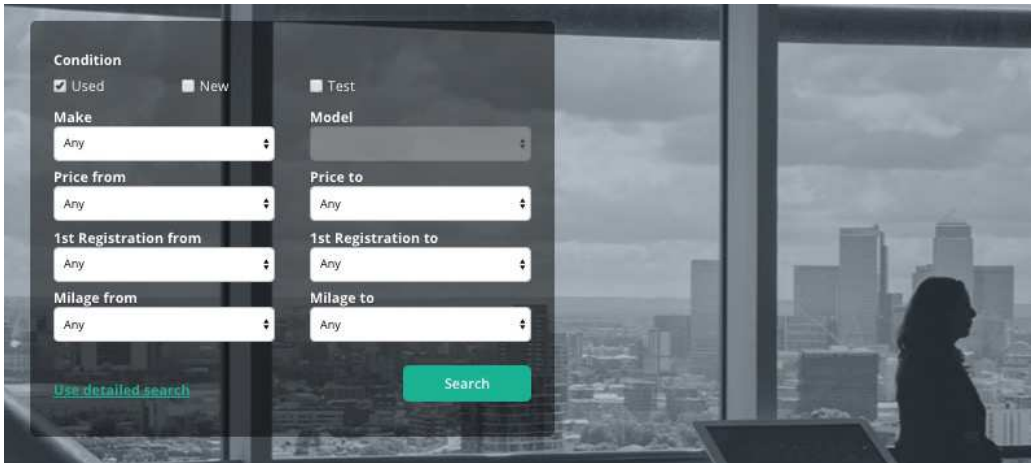
Velika večina migracij izgleda zelo podobno. V njih dodajamo ali odvzemamo polja in nastavljamo datotečne atribute ali pa brišemo oziroma dodajamo celotne tabele.

3.2 Začetna stran

Namen začetne strani je predstavitev naše spletne aplikacije. Najbolj pomemben del je spletni iskalnik, zato je postavljen na prvo mesto. Pod njim se nahaja nekaj stolpcev, ki opisujejo prednosti aplikacije, za njimi je predstavitveni video. Na sredini je podrobno razloženo delovanje storitve in predstavitev glavnih funkcij, na dnu pa so osnovni podatki in kontakti.

Eden izmed glavnih namenov strani je nagovarjanje uporabnika k registraciji in k brskanju med vozili, zato na strani obstaja več gumbov za registracijo. Začetna stran prepoznava ali uporabnik obiskuje stran prvič, ali se vrača. Na podlagi tega se prikažejo različna sporočila.

Glavni del strani na začetni strani aplikacije je iskalnik med vozili (slika 15). Iskalnik razdelimo v pet skupin polj. V vsaki skupini izbiramo eno izmed značilnosti vozila. Polja delimo na ohranjenost vozila, znamko, model, cenovni razpon, čas prve registracije in prevožene kilometre vozila. Na iskalnem obrazcu je tudi povezava do razširjenega iskalnika, ki še ni bil razvit. Razširjen iskalnik bo v svoji končni obliki omogočal iskanje po več parametrih in bo postavljen na svoji podstrani v aplikaciji.



Slika 15: Iskalnik na prvi strani aplikacije

Pomembnejša dela kode za iskalnik se nahajata v sprednjem delu in vključujeta polji z imenoma Make in Model, ki predstavljata znamko in model vozila. Obe polji sta izbirni polji, kar pomeni, da pri vsaki iz seznama izberemo po eno vrednost. Skrajšana HTML koda za polje Make (znamka vozila):

```
<%= select_tag(:make,
        '<option value="*">Any</option>
        <option value="1|mer|Mercedes-Benz">Mercedes-Benz</option>
        <option value="2|vol|Volkswagen">Volkswagen</option>
       —- mankajoča koda —-'.html_safe,
        id: "make-selector", class: "form-control" ) %>
```

Polji sta med seboj povezani tako, da na podlagi izbire v prvem polju prikažemo podatke v naslednjem polju (model vozila). Na primer, če izberemo znamko vozila Mercedes-Benz v prvem polju, se izbere opcija 1. Zato se v naslednjem polju prikažejo podatki, ki spadajo pod opcijo 1. To pa sta modela 190 in 200. HTML kodo za polje ustvari Ruby pomagalnik oziroma značka **select_tag**, v katero podamo parametre. Skrajšana HTML koda za polje Model (model vozila) je prikazana spodaj.

```
<%= select_tag(:model,
        '<option value="*" selected="selected">Any</option>
        <option value="1|*">Any</option>
        <option value="1|190">190</option>
        <option value="1|200">200</option>
        <option value="2|181">181</option>
        —- mankajoča koda —-'.html_safe,
        id: "model-selector", class: "form-control" ) %>
```

Ko uporabnik izbere polja za iskanje in klikne na gumb za iskanje, se izbrani parametri pošljejo v kontroler. Kontroler nato najde vsa vozila, ki ustrezajo vsem kriterijem iskanja. Večino logike za iskanje smo postavili v svojo metodo, ki se imenuje **find_vehicles** (slika 16).

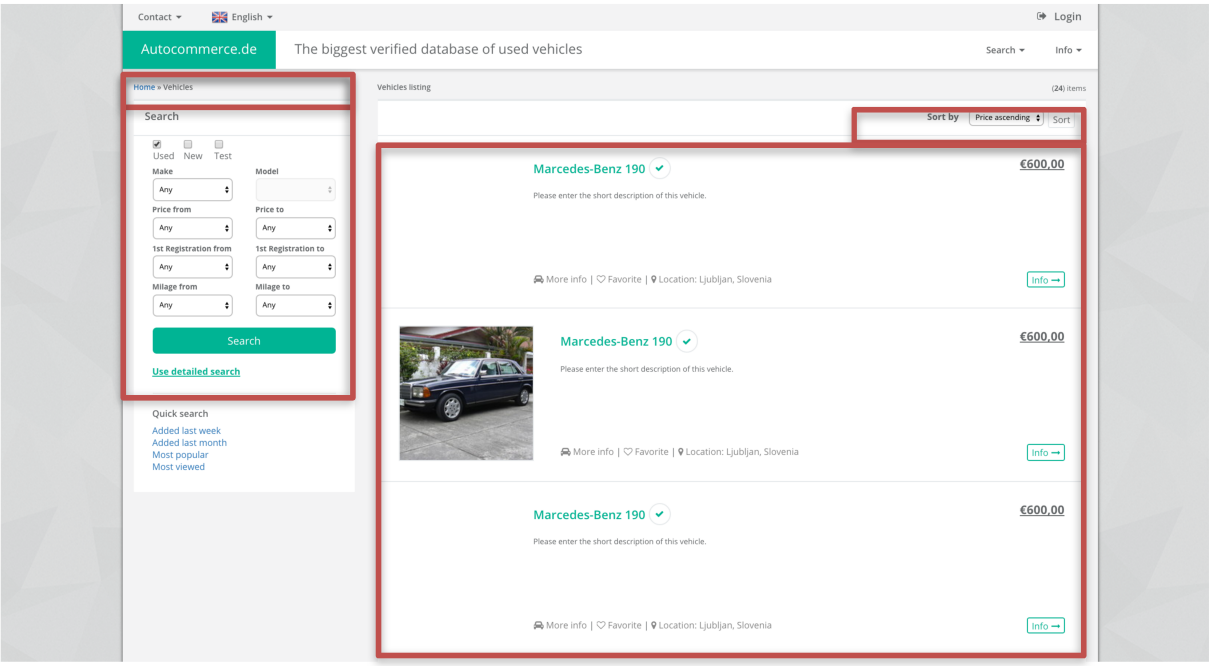
```
6   def find_vehicles
7     #seach method for landing pages
8     make_param = params[:make].split("|")[2] if params[:make].present?
9     model_param = params[:model].split("|")[1] if params[:model].present?
10
11    vehicles = Vehicle.order(:make)
12    vehicles = vehicles.where("condition_new = 't'", params[:condition_new]) if params[:condition_new].present?
13    vehicles = vehicles.where("condition_used = 't'", params[:condition_used]) if params[:condition_used].present?
14    vehicles = vehicles.where("condition_test = 't'", params[:condition_test]) if params[:condition_test].present?
15
16    vehicles = vehicles.where("make like ?", "%#{make_param}") if make_param.present?
17    vehicles = vehicles.where("model like ?", "%#{model_param}") if model_param.present?
18
19    vehicles = vehicles.where("price >= ?", params[:price_from]) if params[:price_from].present?
20    vehicles = vehicles.where("price <= ?", params[:price_to]) if params[:price_to].present?
21
22    vehicles = vehicles.where("registration_year >= ?", params[:registration_from]) if params[:registration_from].present?
23    vehicles = vehicles.where("registration_year <= ?", params[:registration_to]) if params[:registration_to].present?
24
25    vehicles = vehicles.where("milage >= ?", params[:milage_from]) if params[:milage_from].present?
26    vehicles = vehicles.where("milage <= ?", params[:milage_to]) if params[:milage_to].present?
27
28    @number_of_results = vehicles.count
29    vehicles
30  end
```

Slika 16: Metoda **find_vehicles** v kontrolerju, ki vrača vozila

Metoda na začetku preveri, ali sta prisotna parametra za model in znamko. Nato izlušči tekstovne podatke za iskanje in začne preverjati vsak kriterij posebej. Na začetku preveri tip vozila, nato ceno, leto registracije in nato še število prevoženih kilometrov.

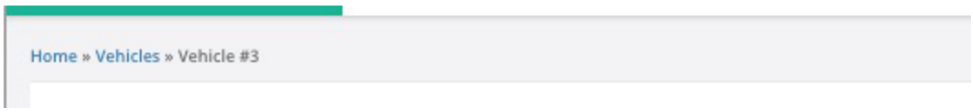
3.3 Brskanje med vozili

Ko izvedemo iskanje, se znajdemo na strani, ki nam prikaže vsa vozila v tabeli (slika 17). V vsaki vrstici je eno vozilo. V vrstici so izpisani podatki, kot so cena, lokacija, znamka ter slika vozila. Ob kliku na vozilo smo preusmerjeni na podstran vozila. Ob brskanju med vozili imamo na levi strani manjši iskalnik, s katerim lahko ponovno iščemo med vozili.



Slika 17: Glavni deli strani na podstrani za brskanje med vozili

Na tej strani smo v zgornji levi kot strani dodali tudi enostavni podmeni, ki deluje kot kazalo globine. To uporabnikom olajša navigacijo tako, da kazalo doda še dodatne informacije o položaju na strani. Kazalo globine je del '**breadcrumbs_on_rails**' paketa, ki olajša delo z enostavnejšimi številčenji pri izpisu podatkov.



Slika 18: Primer kazala globine

Kazalo globine (slika 18) se doda na spletno stran med HTML kodo z ukazom render_breadcrumbs:

```
<%= render_breadcrumbs %>
```

V kontrolerju Vehicles (vozila) pa dodamo dve vrstici kode, s katerima povemo katera je korenska stran (root) ter kateremu podatkovnemu modelu bomo dodajali številčenje. V našem primeru je to podatkovni model Vehicles. Ostalo uredi paket sam.

```
add_breadcrumb "Home", :root_path
add_breadcrumb "Vehicles", :vehicles_path
```

Zdaj se na strani prikazuje kazalo globine, kjer je korenska stran Home (domov).

Če iskalnik pokaže dovolj zadetkov, se na koncu strani pojavi tudi številčenje strani. Številčenje je del '**will_paginate**' paketa.

Slika 19: Primer številčenja število vozil, ki jih izpiše rezultat iskanja.

Številčenje (slika 19) se aktivira v primeru, da rezultat presega nastavljeno število vozil, ki jih lahko prikažemo na eni strani. Številčenje se v pogled strani doda z enostavnim klicem pomagalnika, ki je definiran v paketu. Številčenje dodamo na stran tako, da v metodi index v kontrolerju vozil (Vehicles) dodamo naslednjo kodo:

```
def index
  @vehicles ||= find_vehicles.paginate(per_page: 20, :page => params[:page])
end
```

Ta koda ob izpisu vozil v index pogledu vsebuje tudi funkcijo, ki postavi na stran številčenje. Funkcija sprejema dva parametra. Prvi parameter je število objektov, ki jih želimo prikazati na eni strani. Drugi parameter pa prenaša aplikacijske parametre.

Ostane nam samo še koda, ki jo dodamo na pogled med HTML kodo vozil in je zelo enostavna:

```
<%= will_paginate @vehicles, class: "apple_pagination" %>
```

Sedaj naša aplikacija omogoča številčenje strani. Dodali smo tudi izgled, ki je definiran v CSS razredu apple_pagination.

Na levi strani se nahaja spletni iskalnik. Je kopija glavnega iskalnika, ki se pojavi na začetni strani, večina kode je popolnoma enake. V projekt je dodan kot nova delna stran, ki je definirana kot **_search.html.erb**. To je storjeno zato, ker je omenjeni iskalnik drugačne oblike

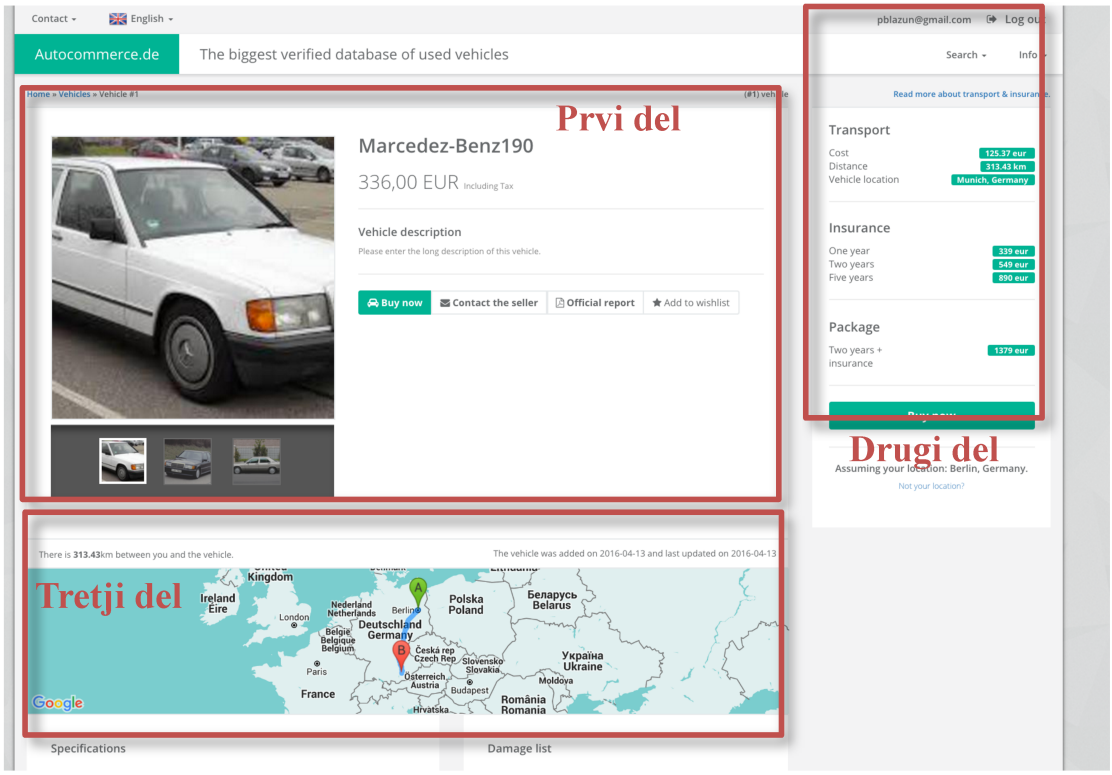
in zato vsebuje drugačno CSS kodo. Vsebuje tudi nekaj manjših sprememb v JavaScriptu. Iskalnik v stran dodamo z ukazom:

<%= render 'search' %>

Delne strani so delčki kode, ki jih večkrat uporabimo na različnih straneh. Takšen pristop je pomemben, če želimo imeti dober pregled nad programsko kodo in več modularnosti. Zato posamezne dele zapakiramo v delne podstrani in jih enostavno vključimo z ukazom **render 'ime podstrani'**.

3.4 Podstran vozila

Prvi del zajema slike ter kratek opis in predstavitev vozila (slika 20). Iz tega dela uporabnik dobi prvi vtis o vozilu na podlagi najpomembnejših informacij. Tukaj so na voljo slike, ki smo jih naložili s pomočjo paketa **Refile**, prikazujemo jih pa s pomočjo vtičnika **lightBoxGallery**. Ta vtičnik ob kliku odpre pojavno okno, ki zatemni ozadje strani in kjer se lahko premikamo med slikami. Drugi del strani (slika 20) je desna stran, ki zajema številke, ki prikazujejo oddaljenost vozila in ceno zavarovanja. Na tem delu strani je zelo izpostavljen gumb "kupi zdaj", ki napeljuje kupca, da kupi vozilo in pri tem izbere še opcijo za prevoz in zavarovanje.



Slika 20: Prikaz podstrani vozila, ki je sestavljena iz treh glavnih delov

Te številke se prikazujejo le v primeru, da je uporabnik registriran in prijavljen v sistem, ker stran uporablja lokacijo in nastavitve posameznega uporabnika.

Tretji del strani (slika 20) se nahaja pod prvim delom in zajema natančnejše podatke o vozilu. Podatki so razdeljeni na zgodovino poškodb vozila ter trenutno stanje, ki je opisano s slikami in tekstom. S tem uporabnik pridobi širšo sliko o vozilu in njegovem stanju in se lažje odloči za nakup.

Celotna podstran je po Bootstrap stolpcih razdeljena v dva dela in loči levi ter desni del. Prvi del zajema 9 kolumn in je na levi strani, na desni strani ostajajo 3 kolumne, kjer se nahajajo številke za transport in zavarovanje.

Slike v levem delu strani izrisujemo z vtičnikom LightBox z naslednjo kodo:

```
<%= form_for(@vehicle) do |f| %>
  <ul class="pgwSlideshow">
    <%  @vehicle.images.each do |i| %>
    <li>
      <a href="<%= attachment_url(i, :file) %>" data-gallery="" >
        <%= attachment_image_tag(i, :file, :fill, 750, 750, format: "jpg") %>
      </a>
    </li>
<% end %>
```

Prvo sliko izrišemo večjo zaradi oblike strani. Nato s pomočjo zanke **for** izrišemo vse slike za podano vozilo v kvadratni velikosti 750 pikslov.

Razdaljo med uporabnikom in vozilom izračuna paket Geocoder. Prvo polje prikazuje razdaljo s številko v kilometrih, naslednje polje pa črkovno mesto lokacije. Razdaljo med dvema točkama izračunamo s spodnjim ukazom.

```
Geocoder::Calculations.distance_between(point1, point2)
```

Pod slikami se nahaja mapa vozila, katero izrisuje paket **gmaps4rails**. V mapi je tudi izrisana najkrajša pot med vozilom in uporabnikom. Za mapo in številke smo naredili delna pogleda. Mapo prikazujemo z uporabo ukaza render:

```
<%= render 'map' %>
```

V datoteki **_map.html.erb** se nahaja večino JavaScript kode, ki poskrbi, da se mapa pravilno izriše v HTML znački z identifikacijo ‘map’, ki se nahaja čez celotno širino 9 kolumn in je visoka 180 pikslov.

```
<div style='width: 100%;'>
  <div id="map" style='width: 100%; height: 180px;'></div>
</div>
```

V kodi preverjamo, ali je uporabnik prijavljen ali ne. Pot izrišemo samo v primeru, da je uporabnik prijavljen. Za to nalogo poskrbi spodnja skripta (slika 21).

```
<% if user_signed_in? %>
 <script>
   /* -- The script will show the route between an user and a vehicle if user is signed in-- */
   var handler = Gmaps.build('Google');
   handler.buildMap({ internal: {id: 'map'}, provider: { styles:
     [{"featureType":"landscape.natural","elementType":"geometry.fill","stylers":[{"visibility":"on"},
     {"color":"#e0efef"}]},{"featureType":"poi","elementType":"geometry.fill","stylers":[{"visibility":"on"},
     {"hue":"#1900ff"},{"color":"#c0e8e8"}]},{"featureType":"road","elementType":"geometry","stylers":[{"lightness":100},
     {"visibility":"simplified"}]},{"featureType":"road","elementType":"labels","stylers":[{"visibility":"off"}]},
     {"featureType":"transit.line","elementType":"geometry","stylers":[{"visibility":"on"},{"lightness":700}]},
     {"featureType":"water","elementType":"all","stylers":[{"color":"#7dcdcd"}]}]
   }}, function(){
     var start_point = <%= Geocoder.coordinates(current_user.city.to_s + ", " + current_user.country.to_s) %>;
     var end_point   = <%=  Geocoder.coordinates(@vehicle.city.to_s + ", " + @vehicle.country.to_s ) %>;
     var directionsService = new google.maps.DirectionsService();
     var directionsDisplay = new google.maps.DirectionsRenderer();

     directionsDisplay.setMap(handler.getMap());
     var request = {
       origin:      new google.maps.LatLng(start_point[0], start_point[1]),
       destination: new google.maps.LatLng(end_point[0], end_point[1]),
       travelMode:  google.maps.TravelMode.DRIVING
     };

     directionsService.route(request, function(response, status) {
       if (status === google.maps.DirectionsStatus.OK) {
         directionsDisplay.setDirections(response);
       }
     });
   });
 </script>
```

Slika 21: Koda, ki poskrbi za izris najkrajše poti med vozilom in uporabnikom

V zgornji skripti paket gmaps4rails ustvari Google zemljevid, ki ima spremenjen izgled s podanimi stilskimi značkami. Nato s pomočjo Geocoder paketa izračuna lokacijo za začetno in končno točko. Preko API vmesnika se nato povežemo na storitev Google zemljevidov preko katere dobimo podatke za izris poti. Če je prenašanje podatkov uspelo, na koncu izrišemo pot na zemljevid z dvema pikama, ki označujeta začetno in končno točko. V primeru, da uporabnik ni prijavljen, se na mapi pojavi samo lokacija, ki označuje mesto vozila.

Desni del strani z številkami smo shranili v svoj podogled, ki se imenuje _numbers.html.erb in ga v podstran vozila vključimo s klicem naslednje kode:

```
<%= render 'numbers' %>
```

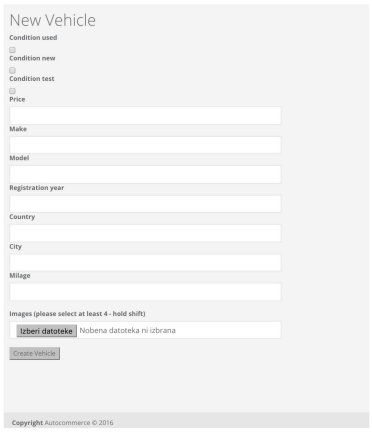
V podogledu številk se nahaja koda (slika 22), ki izpiše ceno prevoza ter razdaljo. Izračuna se tudi zavarovalnina ter možnost nakupa celotnega paketa. Vse te številke se prikažejo v desnem delu spletne stran, kjer se nahaja tudi velik gumb za nakup.

```
<div class="col-lg-6" style="text-align: right;">
  <h3><span style="opacity: 0">.</span></h3>
  <% if user_signed_in? %>
      <span class="label label-primary">
          <%= (Geocoder::Calculations.distance_between(Geocoder.coordinates(current_user.city.to_s + " ," +
           current_user.country.to_s), Geocoder.coordinates(@vehicle.city.to_s + ", "
           + @vehicle.country.to_s ) ) *  0.4).round(2) %> eur
      </span><br>
  <% else %>
      <span class="label label-danger">
          Please log in
      </span><br>
  <% end %>
    <% if user_signed_in? %>
      <span class="label label-primary">
          <%= (Geocoder::Calculations.distance_between(Geocoder.coordinates(current_user.city.to_s +
          " ," + current_user.country.to_s), Geocoder.coordinates(@vehicle.city.to_s + ", "
          + @vehicle.country.to_s ) )).round(2) %> km
      </span><br>
    <% else %>
      <span class="label label-danger">
          Please log in
      </span><br>
    <% end %>
  <span class="label label-primary"><%= @vehicle.city + ", " + @vehicle.country %></span><br>
</div>
```

Slika 22: Koda na podogledu numbers

3.5 Sistem za vnašanje vozil in administracijo podatkov

Vozila se vnašajo na spletnem podnaslovu **/vehicles/new** (slika 23). Ker vozil ne vnašajo uporabniki, ampak pregledovalci vozil, je spletni naslov navadnim uporabnikom nedostopen. Do naslova lahko dostopajo samo uporabniki, ki imajo odobren dostop. Pri vnašanju vozila izpolnimo polja s podatki ter izberemo slike vozila. Vpišejo se tudi podatki iz cenilnega lista. Na koncu se naloži tudi cenilni list v PDF datoteki. Programsko je to izvedeno s pomočjo vgrajenih spletnih obrazcev, ki pošljejo podatke na strežnik, kjer se shranijo.



Slika 23: Spletni obrazec za vnos vozila

Za nalaganje slik in PDF datotek smo uporabili paket Refile, ki je eden izmed novejših paketov in skrbi za nalaganje datotek v RoR aplikaciji. Omogoča hkratno nalaganje večjih datotek, spreminjanje datotek in zelo fleksibilen datotečni sistem. Refile smo dodali v projekt z vključitvijo paketa v Gemfile. Refile zahteva tudi paket mini-magick, ki skrbi za obdelavo slik:

```
gem "refile", require: 'refile/rails', git: "https://github.com/refile/refile.git"
gem "refile-mini_magick"
```

Za tem smo pripravili novo migracijo, ki ustvari nov model za slike in črkovno polje, kamor bomo shranjevali pot do slik. Nato smo v spletni obrazec za vnašanje novega vozila dodali še polje za nalaganje slik:

```
<div class="field">
  <%= f.label "Images (please select at least 4 - hold shift)" %><br>
  <%= f.attachment_field :images_files, multiple: true, presigned: true, direct: true %>
</div>
```

V model Vehicles smo dodali naslednjo kodo:

```
has_many :images, dependent: :destroy
accepts_attachments_for :images, attachment: :file, append: true
```

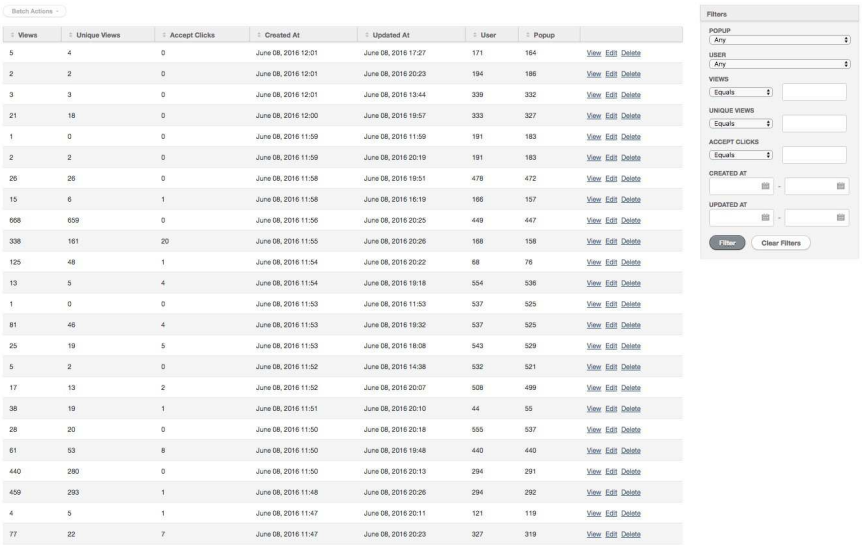
S tem smo naredili asociacijo med podatkovnima modeloma Images and Vehicles v podatkovni bazi. V naslednji vrstici pa smo modelu povedali, da bomo sprejemali datoteke v podatkovni obliki slik (modela Images). Za vse ostalo avtomatsko poskrbi paket Refile.

V konfiguracijskih datotekah paketa lahko spreminjamo pot, kamor se nalagajo datoteke. Mi smo obdržali privzeto konfiguracijo.

Administracija vseh podatkov je na voljo na naslovu **/admin**. Tukaj je obrazec za prijavo v administracijski del. Ob uspešni prijavi nas sistem preusmeri na stran za administracijo, kjer imamo na voljo vse podatke po tabelah, kot so v podatkovni bazi (slika 24). Je grafični vmesnik za nadzor podatkov v podatkovni bazi in se uporablja vsakodnevno za enostaven pregled in urejanje posameznih zapisov. Administracija ja del paketa **ActiveAdmin**, ki ga enostavno naložimo v RoR. Paket je dostopen na spletnem naslovu www.activeadmin.info, kjer je na voljo tudi dokumentacija in začetni primer, s pomočjo katerega vgradimo ActiveAdmin v našo aplikacijo.

Naložimo ga z vpisom Gem paketa v Gemfile datoteko:

```
gem 'activeadmin', github: 'activeadmin'
```



Slika 24: Primer izpisa podatkov in urejevalnika na desni strani v ActiveAdmin

Nato ga enostavno konfiguriramo z ukazom:

```
rails generate active_admin:install
```

Za tem pa migriramo podatkovno bazo in smo z začetno konfiguracijo končali:

```
rake db:migrate
```

Nato za vsako tabelo v podatkovni bazi kreiramo datoteko z imenom podatkovnega modela. Na primer za uporabnike kreiramo datoteko **User.rb** v mapi **autocommerce/app/admin**.

Znotraj datoteke definiramo vsa polja, ki so v tabeli uporabnikov. Lahko definiramo samo tista polja, ki nas zanimajo in jih želimo prikazati in urejati. To storimo z naslednjo kodo:

```
ActiveAdmin.register User do
  index do
    column :id
    column :email
    column :first_name
    column :last_name
    column :created_at
    column :sign_in_count
    column :last_sign_in_at
    column :full_access
    actions
  end
end
```

3.6 Sistem za registracijo in prijavo uporabnikov

Za registracijo in prijavo uporabnikov smo uporabili zelo razvit in široko uporabljen Devise paket [16], ki je odprto dostopen na GitHubu.

Devise je dovršen MVC sistem za uporabniške storitve, ki ga enostavno vključimo v svoj projekt. Uporabljamo ga predvsem za overjanje uporabnikov. Omogoča tudi veliko drugih funkcij, kot so na primer potrjevanje uporabnikov, vodenje evidenc, pošiljanje elektronskih sporočil povezanih z uporabniškimi računi ter zaklepanje. Uporablja in razširja ga tudi naš sistem za administracijo ActiveAdmin.

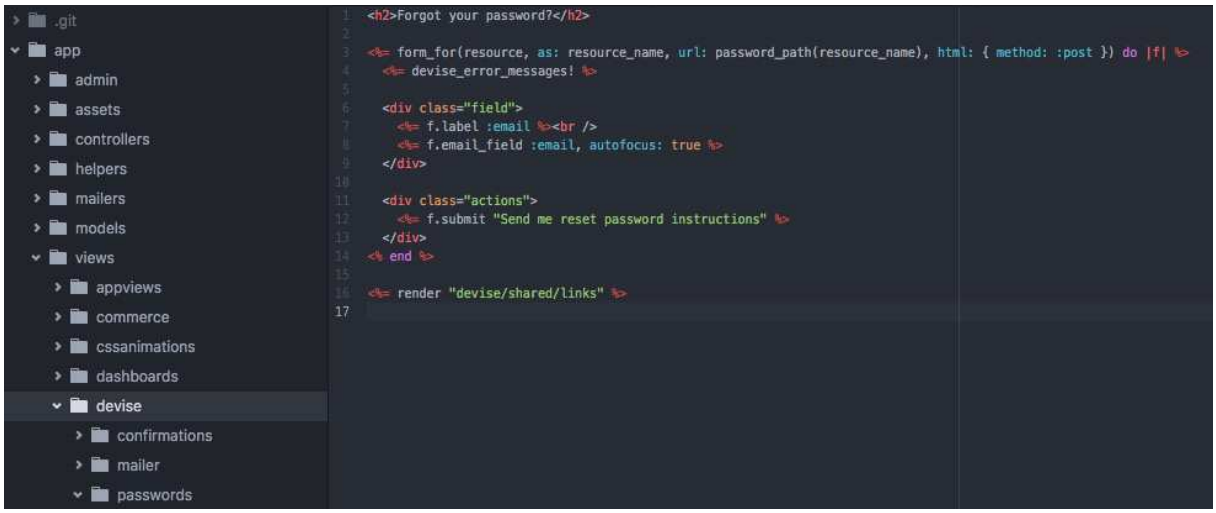
Devise dodamo v projekt z ukazom:

gem 'devise'

Nato zaženemo Devise generator, ki bo ustvaril vse potrebne datoteke in povezave ter nov podatkovni model User, ki služi kot podatkovni model uporabnika. Zaženemo ga z ukazom:

rails generate devise:install

S tem ukazom smo pripravljeni na delo. Generator ustvari tudi uporabniški kontroler in poglede, ki jih lahko prepišemo s poljubno kodo in tako dodamo ali spremenimo logiko delovanja. Na sliki 25 je primer generiranih pogledov in programska koda pogleda za pošiljanje novega uporabniškega gesla.



Slika 25: Primer generiranih pogledov

Za uporabniške storitve nismo imeli posebnih zahtev, zato nismo spreminjali logike paketa Devise. Dodali smo samo dodatno polje, ki shranjuje državo in mesto uporabnika. To smo storili s pomočjo podatkovne migracije, s katero smo vrinili v model User novi polji, ki se imenujeta city (mesto) in country (država). Uporabnik mora ob registraciji vpisati tudi svoje mesto in državo. To je potrebno za prikazovanje izračunov razdalje med uporabnikom in vozilom, ki si ga ogleduje. Če uporabnik ne vnese podatkov se privzeto uporabi lokacija mesta Berlin v Nemčiji. Prilagodili smo tudi izgled vseh podstrani za avtentikacijo uporabnikov. To smo storili tako, da smo že obstoječo logiko vgradili v izgled teme, kar pomeni, da smo prestavili polja, ki jih uporablja Devise za avtentikacijo v nove poglede, ki smo jih dobili skupaj z nakupom aplikacijske teme.

3.7 Ostalo

Sistem komunikacije med uporabniki

Komunikacija med uporabniki znotraj naše aplikacije je potrebna za pošiljanje sporočil med uporabniki. Za komunikacijo med njimi smo uporabili paket MailBoxer [17], ki je na voljo na GitHubu v obliki paketa.

Kljub svoji majhnosti je izjemno močan in fleksibilen. Ima veliko in predano spletno skupnost, kar je ena izmed njegovih prednosti. MailBoxer omogoča pošiljanje zasebnih sporočil in zgodovino uporabnikovih dejanj v aplikaciji. Omogoča tudi pošiljanje elektronskih sporočil glede na dogodke, ki se zgodijo posameznim uporabnikom znotraj aplikacije. Takšno sporočilo se pošlje na elektronski naslov uporabniku, ki ga obvesti ob prejemu novega sporočila v aplikaciji.

Aplikacijska obvestila

Takšna obvestila so namenjena povečanju interakcije z uporabnikom in serviranju dodatnih informacij, ki se prikažejo samo občasno. Prikazujejo se v spodnjem desnem kotu aplikacije. Namenjena so obveščanju posameznih uporabnikov o dogodkih v aplikaciji.

Eden izmed takšnih dogodkov je prejem novega sporočila. Ob brskanju med vozili se sistem uči o zanimanjih specifičnega uporabnika in mu na podlagi podatkov servirajo sporočila o vozilih, ki bi ga lahko zanimala.

Takšna sporočila smo vgradili v aplikacijo s pomočjo JavaScript vtičnika, ki se imenuje Toastr. To je JavaScript knjižnica, ki omogoča implementacijo sporočil.

V glavno aplikacijsko JavaScript datoteko smo podali pot do vtičnika Toastr ter ga tako vključili v aplikacijo. Na podstraneh prikazujemo sporočila tako, da smo na dnu HTML datotek ustvarili skriptne značke <script>, kamor smo dodali skriptno kodo za sporočila.

V enostavni obliki imamo dva JavaScript ukaza, s katerima nastavljamo sporočila. Prvi ukaz je opcijski ukaz, s pomočjo katerega nastavljamo nastavitve sporočil. Tukaj na primer nastavimo čas prikaza sporočila ter barvo.

```
toastr.options = { }
```

V naslednjem ukazu pa vnesemo vsebino sporočila. Primer takšnega sporočila je spodaj, ko uporabnika pozdravimo ob ponovnem obisku aplikacije.

```
toastr.success('Welcome back to Autocommerce!');
```

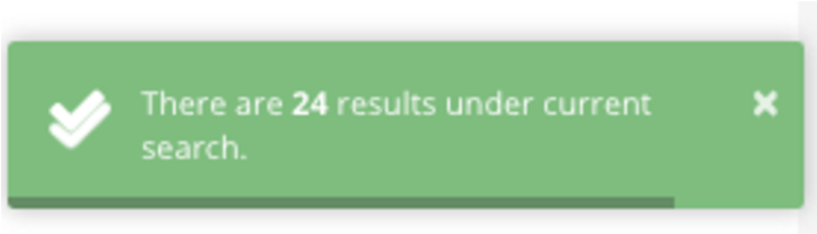
V aplikaciji smo takšna sporočila uporabili zelo enostavno. Če želimo prikazovati dinamično generirana sporočila, moramo povezati Toastr z zadnjim delom aplikacije. To lahko storimo tako, da spremenimo ime pogleda strani iz **index.html.erb** v **index.html.js.erb.**

S tem povemo RoR ogrodju, da bomo tudi v JavaScript kodo dodajali Ruby ukaze. RoR nato avtomatsko ovije celotno JavaScript kodo z dodatno plastjo, kar omogoča pisanje Ruby kode v JavaScriptu. Nato lahko prikažemo sporočila uporabniku z Ruby ukazom. Izmišljena metoda **get_latest_vehicles** izpisuje vozila, ki bi lahko zanimala uporabnika. Ukaz je naslednji:

```
toastr.success('<%= user.get_latest_vehicles %>');
```

V naši aplikaciji je primer takšnega sporočila spodaj (slika 26), ko uporabnika obvestimo, koliko vozil obstaja kot rezultat trenutnega iskanja med vozili.

```
toastr.success('There are <b><%= @number_of_results %></b> results … ');
```



Slika 26: Primer obvestila ob iskanju v spodnjem desnem kotu

Plačilni sistem

Ker je aplikacija še v začetni fazi, v času pisanja diplome še nismo uspeli implementirati plačilnega sistema. Implementiran je samo sporočilni sistem, s pomočjo katerega si kupec in prodajalec pišeta sporočila. Ko pridemo do točke, da bomo začeli razvijati plačilni sistem, bomo uporabili ali PayPalov sistem Braintree ali pa Stripe. Oba sistema omogočata fleksibilno in relativno lahko zasnovo kompleksnejšega plačilnega sistema. Sistema sta na voljo v obliki paketov z obsežno dokumentacijo, ki se jih doda v Ruby on Rails projekt. Braintree in Stripe sta odprto dostopna na GitHubu.

Plačilni sistem bo integriran v aplikacijo tako, da bo uporabnik na začetku nakupa vpisal svojo plačilno kartico, s katere se bo del zneska prenesel takoj ob nakupu, del zneska pa ob prejemu vozila. Zaradi zavarovanja bomo denar vrnili, če kupec z nakupom ne bo zadovoljen. Kupec bo kril le stroške, ki bodo nastali zaradi prevoza in zavarovanja. V primeru, da se nakup zaključi, bo prodajalec avtomobila dobil izplačan znesek prodaje vozila. Kupec bo ob nakupu lahko izbral dostavo ali zavarovanje za vozilo, znesek bo v tem primeru izplačan zavarovalnici oziroma partnerju, ki dostavlja vozila.

Vsebina in prevod

Vsebina spletne strani je za spletne iskalnike prilagojena tako, da v vsakem jeziku izpostavlja besedno zvezo “rabljena vozila”. Vsebina prve strani izpostavlja funkcionalnosti aplikacije in kaj lahko z njo počnemo, ter komu je namenjena.

Uporabnik lahko na vrhu strani izbira med angleškim in nemškim jezikom s pomočjo spuščajočega menija (slika 27). Kasneje bomo dodali tudi ostale jezike. Možnost večjezičnosti je že privzeto integrirana znotraj RoR aplikacije, zato je dodajanje novih prevodov enostavno.

Slika 27: Izbira jezika spletne strani

Prevajamo lahko samo statične tekste, ki se shranjujejo v spremenljivkah. V konfiguracijskih datotekah nastavimo privzeti jezik, ki se prikaže ob prvem obisku.

Obstaja tudi možnost, da vsakemu uporabniku nastavimo jezik glede na lokacijo, iz katere prihaja v aplikacijo. Takšne možnosti v naši aplikaciji nismo uporabili, saj si želimo privzeto uporabljati angleški jezik.

Datoteke s prevodi se nahajajo v mapi **config/locales.** Primer takšne datoteke je **en.yml,** kamor shranjujemo angleške tekste, ki se prikazujejo v aplikaciji ob izbiri angleškega jezika.

Koda spodaj prikazuje primer vsebine v datoteki **en.yml**:

```
"body":{
  "welcome": "Welcome Amelia"
},
"topnav":{
  “welcometext": "You have 42 messages and 6 notifications",
  "logout": "Log out",
},
```

V HTML kodi posameznih podstrani nato dodamo tudi **data-i18n** značke. S tem naredimo asociacijo med posamezno značko in prevodom, ki se nahaja v datoteki s prevodi. Primer takšne značke naslednji:

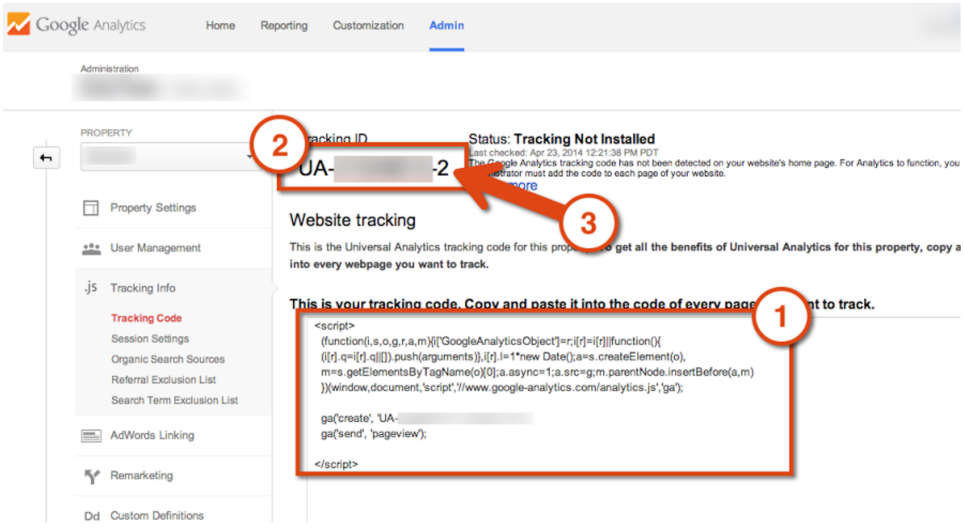
```
<span class="m-r-sm text-muted welcome-message" data-i18n=“topnav.welcometext">
      Welcome to autocommerce!
</span>
```

Če bi želeli imeti prevod v nemškem jeziku, bi ustvarili datoteko de.yml, ter dodali prevod v datoteko:

```
topnav":{
  "welcometext": "Sie haben 42 Meldungen und 6-Benachrichtigungen",
  "logout": "Log out",
},
```

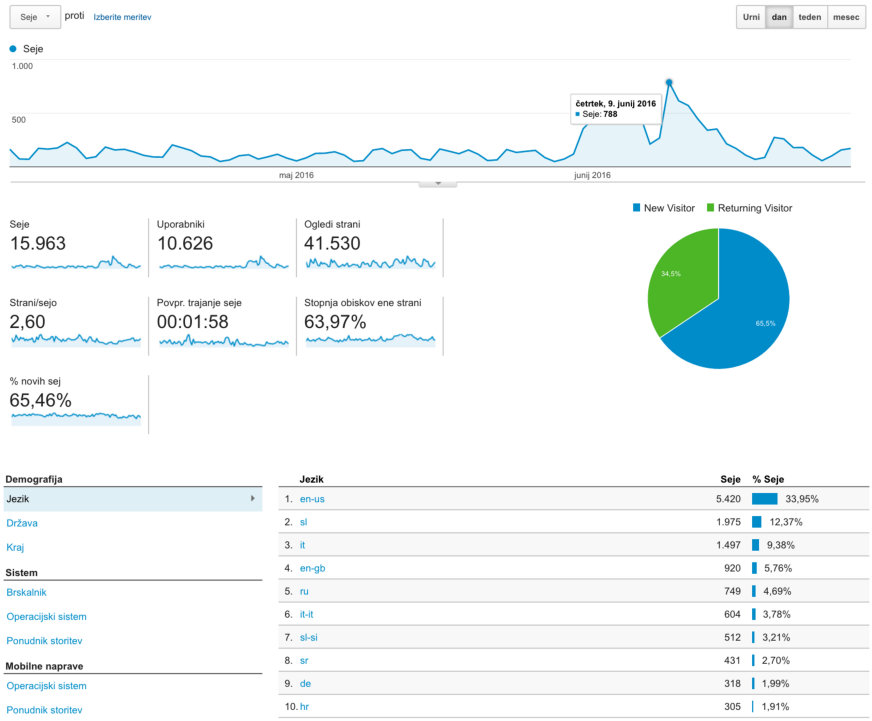
Integracija analitike

V projektu smo uporabili analitični orodji **Google Analytics** in **Mixpanel**. GA smo vgradili tako, da smo v glavo strani dodali skripto, ki se izvede ob nalaganju strani na strani uporabnika.



Slika 28: Pridobivanje kode za integracijo z GA

S to enostavno integracijo GA deluje. Kodo, ki jo moramo vključit v našo aplikacijo, dobimo v storitvi GA, kot je prikazano na sliki 28. Omenjeno kodo smo kopirali v glavo strani. Po tem smo začeli dobivati podatke, ki jih lahko sedaj izrabljamo za posredovanje AdWords oglasov. Podatke vidimo v obliki grafov in številk v nadzorni plošči storitve GA. Primer takšnega prikaza je na sliki 29.



Slika 29: Primer analitičnih podatkov v GA

Integracija z analitičnim orodjem **Mixpanel** [18] je bolj zahtevna, ker ga uporabljamo za spremljanje uporabnikov in njihovega vedenja skozi celotno aplikacijo in na vsaki podstrani. Za ogrodje RoR obstaja paket imenovan mixpanel, ki ga enostavno vključimo v projekt z ukazom:

```
gem 'mixpanel'
```

Nato dodamo gesla in API ključ s katerim povežemo našo stran s storitvijo Mixpanel. Ob uspešni povezavi nas storitev o tem obvesti.

Sedaj nastopi premislek in zasnova koncepta, kako in kaj bomo spremljali. V naši aplikaciji bomo spremljali obiske različnih podstrani in uporabo funkcij. Prav tako bomo spremljali, koliko uporabnikov se registrira in prijavi ter kakšna vozila iščejo.

Integracijo lahko izvedemo s pomočjo kode JavaScript v pogledih ali s kodo Ruby v kontrolerjih. Mi smo se odločili za integracijo s pomočjo Ruby kode v kontrolerjih, zato ker že uporabljamo JavaScript in si želimo obdržati dober pregled nad programsko kodo.

To smo storili tako, da smo najprej v aplikacijski kontroler dodali pomagalnik (angl. helper), ki vedno najprej identificira uporabnika s storitvijo Mixpanel. To smo storili z naslednjim ukazom:

```
before_action :mixpanel_identify
```

ter z metodo spodaj:

```
def mixpanel_identify
    mixpanel.append_identify current_user.email if current_user && !current_admin_user
end
```

Metoda **mixpanel_identify** identificira uporabnika, ampak samo v primeru, da uporabnik ni administrator, saj si ne želimo shranjevati podatkov za administratorja, ker bi s tem popačili realnost podatkov.

Nato v vseh metodah v različnih kontrolerjih, kjer želimo spremljati uporabnika, dodamo naslednjo kodo:

```
mixpanel.append 'track', " -Ime akcije -" unless current_admin_user
```

V njej ustrezno spremenimo -Ime akcije- v smiselno ime funkcije, ki jo želimo spremljati. S to zelo osnovno implementacijo analitičnega orodja Mixpanel začnemo pridobivati podatke v grafe, ki prikazujejo uporabo različnih funkcij in podstrani, ki so sedaj prikazani v grafih in tabelah podobno kot v storitvi GA.

4. Metodologije agilnega razvoja

Agilne metodologije so zelo povezane z današnjim razvojem aplikacij, zato smo jih tudi vključili v našo diplomsko nalogo. Naš projekt je bil razvit s pomočjo agilne metodologije Scrum [19].

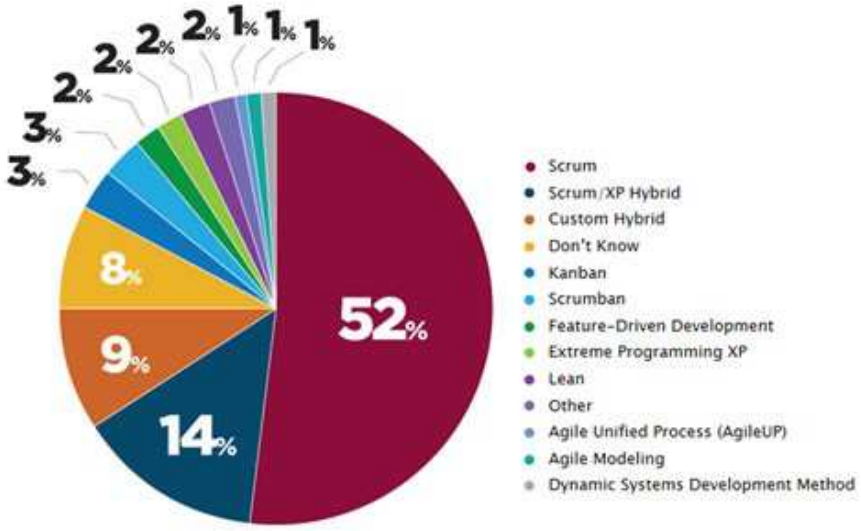
Celotna filozofija razvoja danes bazira na **metodologijah agilnega razvoja** [20] in kulturi zagonskih podjetij. Metodologije so se skozi zgodovine močno spreminjale, predvsem zaradi tehnologij, ki jih uporabljamo za razvoj. Od pojavitve prvih sistemov in metodologij je minilo že kar nekaj časa. Za razliko od njih pa moderne metodologije dopuščajo veliko več svobode, zato je povsem logično, da se z njimi dobro razumejo predvsem mlajše generacije. Agilne metodologije so skupek metod za razvoj programske opreme, po katerih se zahteve in rešitve prilagajajo med sodelovanjem samoorganiziranih in multifunkcijskih skupin. Promovirajo prilagodljivo planiranje, niz dobrih inženirskih praks, hitro dostavo rešitev, nenehne izboljšave in spodbujajo hitro prilagodljivost na sprememb. Najprej so se metode agilnega programiranja imenovale lahke metode (“Lightweight”). Leta 2001 pa so se v mestu Snowbird (Utah, USA) zbrali izkušeni inženirji programske opreme in prvič uporabili ime “agilne metodologije”. Nekateri od njih so kasneje ustanovili neprofitno organizacijo “The Agile Alliance”, katere cilj je promoviranje agilnega razvoja. Avtorji agilne metodologije so ustvarili manifest agilnih metodologij, ki predpisuje principe in postopke, ki bi jih morale upoštevati vse agilne metodologije. Značajsko se razlikujejo od zgodnjih načrtno usmerjenih inženirskih metod. Najočitnejša razlika je vztrajanje pri manj popolnejši dokumentaciji za dani problem. Namesto dokumentaciji se agilne metodologije bolj posvečajo sami kodi kot ključnemu delu dokumentacije. A vseeno je takšno videnje agilnih metodologij površno. Je le posledica mnogo globljih razlik, ki so po Martinu Fowlerju naslednje:

- **agilne metodologije so prilagodljive**
- **agilne metodologije so orientirane k ljudem in ne k procesom**

Predvsem so agilne metodologije pisane na kožo zagonskim podjetjem in srednjim podjetjem, čeprav jih uporabljajo podjetja vseh velikosti. Zaradi hitro spreminjajočih se trendov in trga so postale nujnost v današnjem svetu hitrega napredka. Agilni pristop se ne uporablja samo pri razvijanju programskih produktov, ampak se je razširil v vse industrije in panoge.

Agilne metodologije

Najbolj uporabljena agilna metodologija je metoda Scrum (slika 30). Scrum je izkustven razvoj s poudarkom na prožnosti, prilagodljivosti in ustvarjalnosti. Izvira iz imena Rugby – vrnitev žoge v igro. Pokriva vodenje projekta ali manj natančno voden proces razvoja ter različne aktivnosti. Redefinira naloge vodstva, ki bazirajo na tem, da vodstvo ustvarja okoliščine, ki čim manj motijo delo razvojne skupine. Skupine se organizirajo same.



Slika 30: Prikaz razdelitve uporabe različnih agilnih metod

Samo ogrodje metode Scrum je iterativen inkrementalen proces. V praksi to pomeni, da razvoj poteka v iteracijah, ki si sledijo ena za drugo. Proces lahko razumemo kot neskončnao zanko, znotraj katere razvijamo dodatne funkcionalnosti in popravke izdelka. Zanka je en cikel, ki ponavadi traja teden dni. Vsak dan pa poteka dnevni pregled dela in prilagoditve. Obstaja še nekaj drugih metod agilnega razvoja, ki jih ne bomo podrobno opisali, ker za našo diplomsko nalogo niso pomembne. Te metode so:

- prilagodljiv razvoj programske opreme – ASD (angl. Adaptive Software Development),
- agilne tehnike podatkovnih baz,
- ekstremno programiranje,
- družina metodologij Crystal,
- agilno modeliranje.

Agilne metodologije – principi

Agilne metodologije vsiljujejo zelo malo pravil, med njimi je nekaj principov, ki jim sledijo prav vse metode. V agilne principe spada dvanajst zapovedi, ki si po vrsti sledijo:

1. Najvišja prioriteta je zadovoljiti naročnika s hitro in neprestano dostavo uporabne programske opreme.

2. Spremembe zahtev se spodbujajo, tudi pozneje v razvoju. Agilni proces šteje upoštevanje sprememb zahtev kot primerjalno prednost za naročnika.

3. Dostava delujoče programske opreme naj bo čim pogostejša, od nekaj tednov (zaželjeno) do največ nekaj mesecev.

4. Naročniki in razvijalci naj pri razvoju vsakodnevno sodelujejo.

5. Projekte je treba zastaviti okrog motiviranih posameznikov. Imeti morajo ustrezno okolje in potrebne vire, predvsem pa jim je treba zaupati, da bodo zastavljeno delo opravili.

6. Najprimernejši in najučinkovitejši način za posredovanje informacij o projektu in za projekt je osebni pogovor.

7. Najpomembnejše merilo napredka je količina delujoče programske kode.

8. Agilni procesi predvidevajo nenehni razvoj v okviru projekta. Naročniki, razvijalci in sponzorji morajo biti pripravljeni stalno in na ustrezen način podpirati razvoj v okviru projekta.

9. Nenehna težnja za tehnično odličnost in optimalen načrt razvijajoče programske opreme pospešuje agilnost.

10. Bistvenega pomena je enostavnost – sposobnost maksimiziranja tistega dela razvoja, ki še ni razvit.

11. Najboljša opredelitev zahtev, načrti in programska koda nastanejo pri skupinah, ki se samostojno organizirajo in vodijo.

12. Občasno (ampak redno) mora skupina preverjati, kako bi postala učinkovitejša, in v ta namen sprejeti ustrezne ukrepe.

4.1 Uporaba v praksi

Agilne metodologije so teoretičen skupek idej in principov, ki predstavljajo smernice za razvijanje programskih produktov v hitro spreminjajočem se svetu. V teoriji nam nudijo splošen pregled, kako naj bi danes potekal razvoj. V praksi se stvari bolj zapletejo in niso tako pregledne, prav takrat pa agilne metode priskočijo na pomoč z enostavnostjo in sledenjem preproste logike. Vsako podjetje in tudi vsak človek prilagodi svoj sistem dela drugače, glede na potrebe produkta oziroma storitve, ki jo razvija.

Razvijalci na različnih stopnjah razvoja uporabljajo različne metode, ki jih spoznajo in uporabljajo skozi proces ustvarjanja in so lahko zelo specifične glede na vsak projekt. Zelo pomembno je razumeti, da temeljijo na zagotavljanju svobode za ustvarjalce. Obstaja samo nekaj pravil, ki so zelo logična in sledijo navodilom zdrave pameti. Vsakršnim dodatnim pravilom se želimo izogniti.

4.2 Uporaba na projektu Autocommerce

Sistem dela se je na projektu razvil spontano in v skladu z agilnimi metodologijami. Ker nismo delali skupaj v pisarni, ampak vsak iz svojih prostorov doma ali v službi, smo opravljali Skype klice na dva ali tri dni. Med klici smo se pogovarjali in testirali aplikacijo in podajali mnenja. Slednje je zelo pomembno, saj vsak izmed nas prihaja iz različnega ozadja in doprinese k projektu nekaj drugega. S tem smo sledili 4., 6. in 8. točki iz spiska principov agilnih metodologij. Ekipa je sestavljena iz treh ljudi. Sam sem prevzel predvsem tehnični in razvojni del, druga dva člana pa sta pomagala predvsem v obliki mnenj in pri sestavi uporabniške izkušnje ter pri vzpostavljanju različnih procesov znotraj aplikacije – na primer pri procesu prodaje vozila. Aplikacijo smo vsakodnevno testirali in si zapisovali napake ter izboljšave. Glede na to smo tudi prilagajali izgled aplikacije in načrtovali naslednje funkcije, ki jih bomo razvili. Nove verzije izboljšav so nstajale v enotedenskih ciklih. S tem smo sledili 1., 9. in 11. točki principov agilnih metodologij.

Pomemben poudarek pri celotnem projektu je bil na enostavnosti, hitremu napredku pri razvoju ter komunikaciji.

4.3 Povezava z zagonskimi podjetji

Uporaba agilnih metodologij se je pojavila skupaj s popularizacijo kulture zagonskih podjetij. Agilnost in hiter razvoj sta temelja, s katerim zagonska podjetja ustvarjajo produkte. Ker se takšne metode zelo prepletajo z vsemi ostalimi vidiki, ki jih moramo upoštevati za uspešen razvoj aplikacije oziroma podjetja, bomo v naslednjem poglavju našteli nekaj najpomembnejših

Prikazati želimo miselno širino in izpostaviti stvari, ki so potrebne za medsebojno povezovanje in prepletenost različnih delov kot so razvoj, marketing, dizajn, analitika, komunikacija, ekipa ipd. – vse na podlagi agilnih metodologij.

Motivacija za takšne opise izhaja predvsem iz pomanjkanja določenih znanj na področjih, ki niso vključena v izobraževanje na področju računalništva, a so enako pomembna.

Ekipa

Ekipa je najbolj važen del vsake podjetja. Ekipe so na začetku majhne. V njih na začetku sodelujejo povprečno trije ljudje. Najbolj primeren začetek za zagonsko podjetje, ki deluje na spletu, je trojica “Hustler” , “Programmer” in “Designer”.

“Hustler" (tržnik, menedžer) ustvarja in išče priložnosti za podjetje in ljudi. Takšni ljudje so velikokrat vodje ekip. Odlikuje jih široko razumevanje procesov in produktov. So odlični govorci, katerih odlika je odlična komunikacija.

“Programmer" (programer) je človek, ki skrbi za razvoj produktov in storitev. Je oseba, ki je odgovorna za razvoj in ima širok spekter znanja spletnih in drugih tehnologij.

“Designer" (dizajner) ustvari začetno podobo podjetja, skrbi za preglednost in funkcionalnost uporabniških storitev, skrbi za vse vizualne aspekte storitev in produktov. Zagotavlja funkcionalnost na podlagi vizualnosti in tesno sodeluje s programerjem.

Vsi deli ekipe morejo delovati tako, da medsebojno komunicirajo in si s tem pomagajo na poti k skupnemu cilju. Vsak član ekipe mora biti pripravljen opravljati tudi kakšno delo, ki ga ne mara in v katerem ni najboljši.

Komunikacija in miselna usmeritev

Komunikacija v ekipi in miselna usmeritev vsakega posameznika sta zelo pomembni stvari v zagonskih podjetjih. Ekipa mora vedno stremeti k temu, da lahko vsak aspekt poslovanja konstantno izboljšuje, tudi sebe. Pogosto smo si sami sebi največja ovira na poti k uspehu.

Komunikacija mora biti jasna in odprta ter prepletati vse dele od razvoja do podpore in marketinga. Še bolj pomembno je medsebojno zaupanje. Vsak izmed ljudi dela na svojem področju, ki ga s pomočjo celotne ekipe in zunanjih povratnih informacij izboljšuje na poti k skupnemu cilju.

Problem + rešitev = ideja

Vse se začne z iskanjem problema, ki ga lahko ekipa ali posameznik reši. Idejo, ki se jo splača razvijati, imamo, če identificiramo problem, za katerega imamo rešitev, ki jo je nekdo pripravljen plačati. To je podlaga in začetek vsakega zagonskega podjetja. Večina današnjih idej ni novih, ampak so modifikacije obstoječih ali deli obsežnejših idej, ki tvorijo nove bolj uspešne oblike, ki se ustrezno prilagajajo času. Ideje niso najpomembnejši faktor za uspeh podjetij, kot so prepričani mnogi. Najpomembnejša je ekipa in njihova izvedba ob pravem času s kančkom sreče.

Preverjanje idej

Ko imamo idejo, jo je potrebno preveriti. Preverjamo zato, da jo ustrezno potrdimo preden začnemo v njo vlagati svoj čas in denar. Danes je večina idej predvsem nova kombinacija že obstoječih idej in sistemov. Preverjanje pomeni iskanje povratnih informacij na podlagi predstavitve naše ideje potencialnim kupcem oziroma uporabnikom. Če so povratne informacije dobre in so potencialni uporabniki pripravljeni uporabljati in plačevati za rešitev problema, imamo potrditev ideje. Pri tem pomagajo tudi zunanji mentorji in podjetniki, ki so takšen proces že večkrat izkusili. Za preverjanje ideje ne obstaja en sam dogodek ali dokaz, ampak skupek faktorjev, na podlagi katerih se odločimo, ali je ideja ustrezna za izvedbo.

Inkrementalni razvoj

S preverjanjem ideje se začne razvoj produkta oziroma storitve. Razvoj poteka inkrementalno v manjših korakih. V tedenskih presledkih prihaja do večjih sprememb, medtem ko se vsakodnevno prilagajmo z manjšimi spremembami. Zelo pomembno je, da upoštevamo povratne informacije uporabnikov – s tem pa se učimo, kako uporabniki sprejemajo naš

produkt. Na podlagi tega identificiramo dele produkta, ki jih je potrebno najprej dodati oziroma spremeniti. V agilnem inkrementalnem razvoju na produktu sodeluje vsa ekipa in je eden izmed najpomembnejših delov, ki je hkrati najbolj časovno potraten. Zaradi tega je pomembno, da razvijamo dele, ki so nujno potrebni.

“Acquire, engage, retain"

Je koncept, ki se uporablja za pridobivanje, angažiranje in ohranitev uporabnikov pri spletnih storitvah. To je pomemben koncept, ki skrbi za delovanje celotne spletne storitve. Z njim višamo angažiranost uporabnikov, večamo število novih ter izboljšujemo zadovoljstvo obstoječih. S tem se izognemo temu, da bi uporabniki prenehali uporabljati našo aplikacijo. To je zelo specializiran del, ki potrebuje veliko razumevanja uporabnikov in same aplikacije. Danes je to delo “growth hackerjev”, ki so vešči masovnega pridobivanja uporabnikov, ko aplikacija začne dosegati vedno več uporabnikov na svetovnem spletu.

Podpora in učenje uporabnikov

V vsaki fazi projekta je potrebna uporabniška podpora, s katero nudimo pomoč uporabnikom pri delu z aplikacijo. Podpora je najpomembnejši del marketinga spletnih aplikacij in s pravilno komunikacijo pri podpori lahko povečamo profit podjetja. V procesu razvoja aplikacije razrešujemo težave in se učimo pomembnih lekcij, s katerimi se bodo srečevali naši uporabniki. Težave nato sistematično odpravljamo, dokler se nehajo pojavljati. Prav tako pomembno je strateško podajanje znanja, ki je potrebno za predajo našim uporabnikom. Namen takšne predaje je povečanje znanja in zaupanja uporabnikov. Vse to pa vodi k pripravljanju uporabnikov na zahtevnejšo uporabo aplikacije ali storitve, ki vodi k povečanju dobičkonosnosti.

Zunanje storitve in avtomatizacija

Zagonska podjetja nimajo na voljo mnogo ljudi, zato je pomembno, da čim več delov aplikacije avtomatiziramo z zunanjimi storitvami. Danes to pomeni prepletanje plačilnih, e-mail, analitičnih, podpornih in marketinških sistemov, ki skrbijo, da avtomatično pridobivamo nove uporabnike in z njimi komuniciramo. V večini primerov za to uporabljamo plačilne in e-mail sisteme, ker so preveč zapleteni, da bi jih sami razvijali. Zato se poslužujemo zunanjih storitev, s pomočjo katerih jih implementiramo v naš produkt.

V vseh delih podjetja težimo k avtomatizaciji procesov. Danes lahko to storimo z uporabo digitalnih storitev na svetovnem spletu (slika 31). Na splošno iščemo rešitev, ki omogoča

popolno avtomatizacijo problema in potrebuje čim manj človeškega vmešavanja. S tem se izognemo dodatnemu zaposlovanju, stroškom in časovnim izgubam.

Slika 31: Avtomatizacija procesov s pomočjo zunanjih storitev

Širjenje in optimizacija

Ko imamo učinkovito ekipo in produkt, ki deluje, je čas za razširitev na globalne trge. Tukaj nam lahko močno pomagajo že uveljavljena podjetja in zunanji partnerji, ki delujejo v podobnih panogah. Širitev pomeni novo zaposlovanje, širjenje razvoja aplikacije ter fizična prisotnost v več glavnih mestih po svetu.

Pomembna je tudi večna optimizacija vsakega dela storitve, saj s tem pridobivamo na produktivnosti in povečujemo dobiček. Priporočeno je, da se optimizacija izvaja na čim višjem možnem nivoju na vsaki točki projekta, ter da navodila prihajajo od vodij in se razširjajo navzdol. Zelo splošni in najbolj pogosti primeri optimizacije so nižanje nepotrebnih stroškov, povečanje produktivnosti ter izboljševanje prodajnega postopka.

4.4 Priporočljiva literatura

Obstaja mnogo knjig, ki še bolj poenostavijo in razširjajo naše razumevanje hitrega razvoja in testiranja idej in bazirajo predvsem na zgodbah ljudi, ki so takšne stvari izkusili. Takšne knjige posredujejo predvsem miselno usmeritev, ki nas pripravi do tega, da smo sposobni slediti principom, ki jih narekujejo agilne metodologije. Nekaj takšnih je navedenih spodaj. Dodani so kratki opisi s pregledom njihove vsebine, katerih namen je motivirati bralca.

- **The Lean Startup** (Eric Ries, izdana leta 2011) [21]

The Lean Startup je ena izmed novodobnih klasik, ki se dotikajo teme modernih principov podjetništva in zagona produktov. Temelji na uporabi vitkih oziroma agilnih pristopov. Eric Ries definira zagonsko podjetje (angl. startup) kot organizacijo, ki se trudi ustvariti nekaj novega, v okoliščinah, ki so skrajno negotove. Knjiga pomaga podjetjem preveriti svojo vizijo in najti način, da se prilagodijo, še preden bo prepozno.

- **Crossing the Chasm** (Geoffrey A. Moore, izdana leta 2006) [22]

V knjigi Corssing the Chasm, ki jo je napisal Geoffrey Moore, ki je eden izmed boljših tehničnih in komunikacijskih gurujev, opiše, kako se znebiti starih marketinških idej in narediti prostor za nove. Vsebuje ogromno zgodb in “how to” principov.

- **Permission Marketing** (Seth Godin, izdana leta 1999) [23]

Permission Marketing je knjiga, ki jo je napisal legendarni pisec Seth Godin, ki je znan po tem, da med seboj prepleta področja psihologije, marketinga, oglaševanja, razvoja in podobnih. V knjigi raziskuje psihologijo marketinga in kako pomembno je, da danes pridobimo dovoljenje kupcev, preden jim želimo prodajati svoje produkte ali storitve. Tradicionalno oglaševanje temelji na iskanju pozornosti. Naj bo to TV reklama, ki prekine najljubšo oddajo, ali telefonski klic, ki prekine družinski obrok. Seth Godin imenuje takšen način oglaševanja "Interruption Marketing" in meni, da ta način ni več učinkovit. V svoji knjigi predlaga nov pristop, kjer ciljni uporabnik sam sprejme oglas.

- **Rework** (Jason Fried & David Heinemeier Hansson, izdana leta 2010) [24]

Rework knjiga redefinira delo. Pokaže drug, boljši in lažji način za uspeh v podjetništvu. Ob prebiranju knjige boste ugotovili, da so plani dejansko škodljivi, zakaj investitorji niso potrebni, zakaj spremljanje konkurence ni potrebno. Knjiga prikaže, zakaj v resnici potrebujete precej manj, kot mislite.

- **Execute** (Josh Long, izdana leta 2012) [25]

Je knjiga, ki predstavi način za zelo hiter razvoj programskih produktov. Temelji na izvedbi (angl. execute). V poglavjih je je predstavljen način dela, ki je osebna različica agilnih metodologij.

5. Zaključek

Na začetku diplomske naloge smo opisali aplikacijo ter jo na kratko predstavili. Pregledali smo tudi tehnologije, s katerimi smo razvili spletno aplikacijo, ter utemeljili njihovo uporabo. Zajeti so programski jeziki, ogrodja, strežniške tehnologije in zunanja orodja. Namen takšnega pregleda je pridobivanje širše slike o razviti spletni aplikaciji ter obseg vseh orodij in storitev, ki jih uporablja. Tako lažje razumemo delovanje in razvoj aplikacije.

V nadaljevanju smo opisali koncept aplikacije, poslovni načrt in nekaj začetnih korakov za pridobivanje baze vozil in uporabnikov. Razvili smo spletni projekt Autocommerce.de kot primer aplikacije, ki jo lahko razvijemo s pomočjo omenjenih spletnih tehnologij, zunanjih orodij ter z metodami agilnega razvoja.

Kot temelj aplikacije smo uporabili Ruby on Rails ogrodje, s katerim lahko zelo hitro razvijamo s pomočjo generatorjev. Takšen razvoj je predvsem primeren za zagonska podjetja, ki želijo uporabiti agilen razvoj, s pomočjo katerega je možno hitro prilagajanje potrebam uporabnikov. Podrobneje smo opisali dele, iz katerih je aplikacija sestavljena, in potek razvoja z delčki pomembnejše kode. Izpostavili smo pomembnejše dele ter predstavili nekaj osnovnih korakov integracije z analitiko.

Nazadnje smo opisali še metode in principe hitrega razvoja ter najpomembnejše stvari pri razvoju aplikacije. Iz uspešnih aplikacij se lahko razvijejo tudi podjetja, zato smo vzpostavili vzporednice in opisali najpomembnejše stvari pri vzpostavljanju zagonskega podjetja. Dodali smo tudi pomembno tujo literaturo, ki se močno navezuje na zadnje poglavje in na celotno diplomsko nalogo. Na splošno smo zaključili z metodami agilnega razvoja, zato ker so usmerjene tako, da iz čim manjšega vložka ustvarimo čim več. Prav na takšen način danes delujejo zagonska podjetja in na takšen način je bila izdelana tudi naša aplikacija Autocommerce.

Literatura

[1] (2016) INSPINIA Responsive Admin Theme. Dostopno na: https://wrapbootstrap.com/theme/inspinia-responsive-admin-theme-WB0R5L90S

[2] (2016) Ruby file uploads, take 3. Dostopno na: https://github.com/refile/refile

[3] (2016) Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web. Dotopno na: http://getbootstrap.com/

[4] (2016) Ruby on Rails. Dostopno na: http://rubyonrails.org/

[5] (2016) MVC Arhitektura. Dostopno na: http://www.ditea.si/sl/tehnologije/

[6] (2016) Slika: Model–View–Controller – MVC. Dostopno na: http://www.ditea.si/sl/tehnologije/

[7] (2016) Usage of web servers for websites. Dostopno na: https://w3techs.com/technologies/overview/web_server/all

[8] (2016) Spletni strežnik Apache. Dostopno na: http://www.apache.org/

[9] (2015) Nginx. Dostopno na: https://nginx.org

[10] (2008) Diplomska naloga: Zoran KREBS, Varnost podatkovne baze SQL. Dostopno na: https://dk.um.si/Dokument.php?id=6608&lang=slv

[11] (2016) Delivering your transactional and marketing email through one reliable platform, SendGrid. Dostopno na: https://sendgrid.com

[12] (2016) SSH Communications Security: SSH Key Management. Dostopno na: http://www.ssh.com/

[13] (2016) How to use FTP. Dostopno na: http://web.stanford.edu/group/ritspub/Documents/MacTools/ftp.html

[14] (2016) What Is SSL (Secure Sockets Layer) and What Are SSL Certificates? Dostopno na: https://www.digicert.com/ssl.htm

[15] (2016) Slack: Be less busy. Dostopno na: https://slack.com/

[16] (2016) Devise - Flexible authentication solution for Rails with Warden. Dostopno na: https://github.com/plataformatec/devise

[17] (2016) A Rails gem to send messages inside a web application. Dostopno na: https://github.com/mailboxer/mailboxer

[18] (2016) Mixpanel, Mobile Analytics. Dostopno na https://mixpanel.com

[19] (2010) Andraž Cej, Agilni razvoj programske opreme po metodologiji scrum. Dostopno na: http://eprints.fri.uni-lj.si/1203/1/Cej_A._-UN.pdf

[20] (2013) K. Schwaber, J. Sutherland, The Scrum Guide. Dostopno na: http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf

[21] (2016) The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Dostopno na: https://www.amazon.com/Lean-Startup-Entrepreneurs-Continuous-Innovation/dp/0307887898

[22] (2016) Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers. Dostopno na: https://www.amazon.com/Crossing-Chasm-Marketing-High-Tech-Mainstream/dp/0060517123

[23] (2016) Permission Marketing: Turning Strangers into Friends and Friends into Customers. Dostopno na: https://www.amazon.com/Permission-Marketing-Turning-Strangers-Customers/dp/0684856360

[24] (2016) Rework. Dostopno na: https://www.amazon.com/Rework-Jason-Fried/dp/0307463745

[25] (2016) Execute - Acting on Ideas Immediately When Inspired Rather Than Following the Normal Rules. Dostopno na: https://www.amazon.com/Execute-Acting-Immediately-Inspired-Following/dp/0988578603